INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN: TELEMÁTICA

PROYECTO FIN DE CARRERA

EXTENSIÓN FIREFOX PARA LA EDICIÓN GRÁFICA DE APLICACIONES BASADAS EN CHICKENFOOT

Autora:
Beatriz López Moreno
Tutor:
Norberto Fernández García
Directores:
Eduardo Martín Rojo
Norberto Fernández García
Mayo de 2011

UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN:
TELEMÁTICA

PROYECTO FIN DE CARRERA

EXTENSIÓN FIREFOX PARA LA EDICIÓN GRÁFICA DE APLICACIONES BASADAS EN CHICKENFOOT

Agradecimientos

Este proyecto marca el final de la carrera que parecía que nunca iba a llegar, pero que tras varios años de estudio y esfuerzo, puedo decir que ha merecido la pena.

Por eso que quiero hacer este agradecimiento por todo el apoyo sin el cual no podría haber llegado a donde estoy ahora.

En primer lugar a mi familia, gracias por la educación y los valores que me han infundido, los cuales han hecho más sencillo mi paso por la Universidad.

A mi novio Rober, gracias por la comprensión y los ánimos, has sido mi vía de escape en los malos momentos.

También quería agradecer a mis dos tutores, Norberto y Eduardo, el tiempo prestado, la paciencia y el apoyo, sobre todo muchas gracias por el pequeño empujón para lograr terminar el proyecto.

Y por supuesto a los compañeros y amigos de estos largos años de vida universitaria, en especial a Ana, Carlos, Jesús, Marce, Marta, Pao, Raquel y Sergio, han sido muchas las horas de estudio, prácticas, exámenes y agobios, pero también han sido muchos los buenos momentos, viajes, salidas y todo el tiempo compartido. Ha merecido la pena conoceros a todos.

Resumen

Actualmente la web consiste en un sistema de distribución de información formado por millones de páginas que ofrecen una información y funcionalidad concreta para la que han sido diseñadas.

Un usuario de la web realiza cada día múltiples consultas rutinarias a distintas páginas para, por ejemplo, obtener información sobre el tiempo, actualizar el twitter, leer el correo, etc. Es por eso que surge la necesidad de poder personalizar un sitio web, de manera que sea posible reunir información de varias páginas en una, eliminar características que no necesitamos e incluso automatizar determinados procesos como rellenar formularios o acceder a una aplicación de correo. De esta manera se consigue que la navegación sea más rápida y productiva.

En el presente proyecto, se desarrolló una extensión para el navegador *Mozilla Firefox* que permite la creación de aplicaciones para personalizar la web mediante el lenguaje *Chickenfoot*.

Este lenguaje permite crear *scripts* para alterar el comportamiento de los sitios web y hacer que las páginas respondan de diferente manera, siendo posible así cambiar el diseño y el comportamiento del código en una web. *Chickenfoot* provee comandos sencillos permitiendo la modificación de páginas web, sin necesidad de conocer su estructura interna.

La extensión desarrollada permite ejecutar los *scripts* creados, directamente al introducir la URL sobre la que se ejecutarán en el navegador *Firefox*. Además, se ha querido facilitar la tarea al desarrollador permitiéndole crear los *scripts* mediante una interfaz gráfica de creación de diagramas de flujo.

Abstract

Nowadays, the world wide web consists of an information distribution system that comprises millions of pages that offers the information and specific functionality for which they have been designed.

The web users make every day numerous routine queries to different pages for example, to obtain information about the weather, to update their twitter, to read their e-mail inbox, etc. For that matter emerges the need to personalize the web site, so that it is possible to gather the information of some pages in one, to delete some features we don´t need, and even to automatize some processes like filling forms or logging in an e-mail application. In this way, we get navigation to be faster and more productive.

In the present project an extension for the *Mozilla Firefox* that permits the creation of applications with the purpose of personalizing web sites by using Chickenfoot language, has been developed.

Chickenfoot language makes it possible to develop scripts to change the websites´ behaviour and make pages respond on a different way, making it possible to change the design of the page and the code´s behaviour in a web. Chickenfoot provides simple commands to use without the need to know the internal structure of a page.

The developed extension, allows to execute the written scripts directly by introducing the URL in the address bar of the navigator. Besides, we wanted to make the task easier to the developer, allowing him to write the scripts through a graphic interface for creating flowcharts.

Índice

1. Introducción 2
1.1. Motivación del proyecto 2
1.2. Objetivos 3
1.3. Estructura de la documentación 4
2. Estado del arte 6
2.1 XML 6
2.1.1 XML: Estructura 6
2.1.2 Características de XML 7
2.1.3 Aplicaciones basadas en XML 7
2.2 HTML 5 8
2.2.1 Novedades de HTML versión 5 8
2.2.2 Canvas 9
2.3 CSS (Cascading Style Sheets) 11
2.3.1 CSS: Funcionamiento 11
2.4 El lenguaje JavaScript 13
2.4.1 Cómo incluir código JavaScript 13
2.5 El Navegador Mozilla Firefox. 14
2.5.1 Drag & drop 15
2.6 XUL 16
2.6.1 Tipos de Aplicaciones XUL 17
2.7 Desarrollo de una extensión Firefox. 19
2.7.1 El archivo install 19
2.7.2 Paquetes chrome y Chrome URIs (Uniform Resource Identifier) 20
2.7.3 El fichero chrome.manifest 21
2.8 Chickenfoot 22
2.8.1 Características de Chickenfoot 23
3. Requisitos de Diseño 24
4. Diseño de alto nivel 27
4.1 Acceso a la aplicación desde el navegador. 27
4.2 Descripción de la extensión. 27
4.3 Creando un diagrama. 30
5. Implementación 31

5.1 La extensión ChickenfootFlowChartEditor 31
5.1.1 Estructura de la extensión. 31
5.1.2 Fichero install.rdf 33
5.1.3 Fichero chrome.manifest 34
5.1.4 Contenido (content) 35
5.1.5 Aspecto (skin) 39
5.1.6 Configuración regional (locale) 40
5.1.7 Capas XUL 41
5.1.8 Versiones de Firefox válidas 43
5.1.9 Cómo crear el paquete de instalación XPI 43
5.1.10 Aplicación final 44
6. Pruebas 47
6.1 Instalación de la extensión Chickenfoot 47
6.2 Administrador de diagramas 47
6.3 Editor de Diagramas 48
6.4 Aplicaciónes finales de ejemplo. 49
7. Conclusiones 51
8. Manual de instalación 53
A.1 Instalación de la extensión ChickenfootFlowChartEditor 53
A.2 Instalación de la extensión Chickenfoot 55
9. Manual de usuario 58
B.1 Administrador de diagramas. 58
B.2 Editor de diagramas. 60
B.3 Creando un diagrama 62
10. Planificación 68
11. Acrónimos 72
12. Bibliografía 73

Capítulo 1

Introducción

1.1. Motivación del proyecto

Actualmente la web es un sistema de distribución de información basado en millones de páginas que nos ofrecen información mediante texto, imágenes, vídeos u otros contenidos multimedia, y a través de las cuales se realiza la navegación mediante hiperenlaces.

Aunque muchas páginas web ofrecen aplicaciones, éstas no dejan de ser páginas "estáticas" que nos ofrecen cierta información y una funcionalidad limitada.

En muchas de ellas se pierde mucho tiempo ya que para acceder a la información que nos interesa hay que seguir varios pasos como registrarse, elegir unas opciones, etc.

Además la funcionalidad que nos ofrecen la determina su desarrollador por lo que una web puede ofrecer información que no es interesante y sin embargo faltar otra que para nosotros sí lo es.

Es por ello que surge la necesidad de poder personalizar un sitio web, modificar su apariencia, reunir información de varias páginas en una, quitar características que no utilizamos y automatizar ciertos pasos haciendo que la navegación sea más rápida y productiva.

Así como los programas ofrecen cierta libertad para personalizar su configuración y apariencia, se intenta aplicar la misma idea a la web.

Hay varias aplicaciones como *GreaseMonkey* que ya permiten realizar esto mediante código *JavaScript,* que se ejecuta directamente sobre las páginas web a personalizar, sin embargo no es una tarea sencilla ya que requiere conocer la estructura de la página.

Es por ello que en el presente proyecto se ha querido facilitar la tarea de personalizar la web, mediante diagramas de flujo y un lenguaje que nos permite modificar las páginas sin conocer su estructura y con comandos sencillos e intuitivos: el lenguaje *Chickenfoot* desarrollado en el *Massachusetts Institute of Technology*.

En concreto, se va a desarrollar una extensión para el navegador *Firefox* de forma que se integre un editor de diagramas de flujo que permita la realización de una aplicación para modificar las páginas web*,* haciendo uso de los comandos que nos proporciona el lenguaje *Chickenfoot*.

1.2. Objetivos

El principal objetivo marcado al inicio de este proyecto, era el desarrollo de una herramienta que permitiera personalizar las páginas web mediante el uso de un lenguaje que nos ofreciera esta posibilidad.

En concreto, la aplicación a implementar consistirá en una extensión para el navegador *Mozilla Firefox,* que ofrezca al desarrollador una interfaz gráfica que le permita, mediante la edición de diagramas de flujo, la generación de una aplicación final para personalizar las páginas web.

Se ha pretendido ampliar las posibilidades que ofrecen aplicaciones como *Greasemonkey* añadiendo la función de generar un *script* final de manera que el desarrollador tuviera la visión de un diagrama de flujo a la hora de crear nuevas aplicaciones, y que un usuario final sin conocimientos en el desarrollo de aplicaciones, fuera capaz de descargar las aplicaciones e instalarlas en su sistema de una forma sencilla.

La extensión a implementar debía ofrecer las siguientes funcionalidades:

- **Editor de diagramas de flujo**: que será capaz de generar el código final, que se ejecutará sobre una página *web* para su personalización.

- **Administrador de diagramas**: que permita la creación, modificación y eliminación de los diagramas en el sistema.

- Mecanismo de **importación/exportación** de aplicaciones finales: que permita a un usuario sin conocimientos de desarrollo de aplicaciones, la instalación en el sistema de diagramas generados previamente.

Para poder implementar estas funcionalidades, se han utilizado las tecnologías XUL, *JavaScript*, CSS, HTML 5 (necesarias para el desarrollo de extensiones para *Firefox*) y para la realización de las aplicaciones finales, *Chickenfoot*.

1.3. Estructura de la documentación

Este documento se encuentra dividido en diferentes capítulos en los que se tratan todos los aspectos relacionados con el presente proyecto. Además se incluyen tres apéndices.

Capítulo 1: Introducción.

Se trata del capítulo introductorio en el que se detallan cuáles son las motivaciones iniciales y los objetivos que se marcaron para la realización del proyecto.

Capítulo 2: Estado del arte

En este capítulo se explican brevemente en qué consisten las tecnologías utilizadas durante el desarrollo del presente proyecto. Se verán aspectos sobre el navegador *Mozilla Firefox*, el desarrollo de extensiones para dicho navegador, las nuevas características de HTML 5 y el lenguaje *Chickenfoot*.

Capítulo 3: Requisitos de diseño

En este capítulo se enumeran los distintos requisitos iniciales que debía cumplir el diseño del proyecto.

Capítulo 4: Diseño de alto nivel

En este capítulo se detallarán los objetivos del diseño de alto nivel del proyecto, cumpliendo los requisitos enumerados en el capítulo anterior. Se van a describir las distintas ventanas que forman la extensión y las operaciones que puede realizar un usuario mientras está utilizando la aplicación.

Capítulo 5: Implementación

En este capítulo se va a describir con mayor detalle cuál ha sido la implementación llevada a cabo para la realización del proyecto. Se va a profundizar sobre cómo se han realizado cada una de las partes que engloba el desarrollo completo. Para ello se incluirán partes de código, indicando los ficheros generados y su contenido, y se detallará exclusivamente lo utilizado en este proyecto.

Capítulo 6: Pruebas

En este capítulo se expone el plan de pruebas utilizado para intentar detectar errores en la aplicación desarrollada y comprobar el cumplimiento de los requisitos iniciales. Para ello se van a enumerar las pruebas específicas que han sido realizadas.

Capítulo 7: Conclusiones

En este capítulo se analizan los resultados obtenidos en el desarrollo del software del proyecto, las dificultades al realizar éste y se citan las conclusiones obtenidas.

Anexo A: Manual de instalación

En este apéndice se describen los pasos que tiene que seguir un usuario para conseguir instalar la extensión en el navegador *Firefox*. La explicación se acompañará de imágenes aclarativas del proceso.

Anexo B: Manual de usuario

En este apéndice se detalla el funcionamiento básico de la aplicación desarrollada. De modo que los usuarios puedan entender fácilmente para qué sirve cada parte de la interfaz gráfica, y cómo se debe utilizar. Del mismo modo, se acompañará a la explicación de imágenes aclarativas del proceso y de un sencillo ejemplo de uso.

Anexo C: Planificación

En este anexo, se muestra la planificación llevada a cabo, los materiales necesarios y el coste detallado del proyecto.

Capítulo 2

Estado del arte

En este capítulo se pretende dar una vista general de las distintas tecnologías utilizadas en el ámbito del presente proyecto de fin de carrera; desde el desarrollo de una extensión para el navegador *Mozilla Firefox* hasta el lenguaje *Chickenfoot* utilizado para implementar los *scripts* finales.

2.1 XML

XML (Extensible Markup Language) [1], es un metalenguaje basado en etiquetas desarrollado por el W3C. Es una simplificación de SGML (*Standard Generalized Markup Language*) especialmente orientada a Internet que permite definir fácilmente nuevos lenguajes (*Extensible*), permitiendo que estos lenguajes sean legibles por humanos y fáciles de procesar mediante programas.

2.1.1 XML: Estructura

Un documento XML comenzará con la declaración de éste, indicando la versión.

```
<?xml version="1.0"?>
<libro>
<titulo> Cien años de soledad </titulo>
<disponible tiempo="24" unidad="horas"/>
<autor> Gabriel García Márquez </autor>
<formato> Rústica </formato>
<publicacion>1967 </publicacion>
<precio cantidad="9.99" moneda="euro"/>
<descuento cantidad="5"/>
<enlacelibro href="/exec/ISBN/84-473-0619-4"/>
</libro>
```

Después se definen los elementos que están formados por:

- Etiqueta de comienzo. Ej.: <titulo>
- Todo lo que se halle en medio (*character data*). Ej.: Cien años de soledad
- Etiqueta de fin. Ej.: </titulo>

Las etiquetas (*markup)* deben comenzar por una letra o guión bajo (_), no puede tener espacios en blanco y no tienen longitud máxima.

Todos los documentos XML requieren un elemento raíz para ser bien formados. El elemento raíz (o etiqueta del documento):

- Debe estar precedido por el prólogo (declaración XML + DTD si lo hubiera)
- Debe incluir el documento por completo (nada fuera del elemento raíz)
- Debe emparejarse con la declaración del DTD (para ser válido)

Para definir un fichero XML correctamente, éste debe cumplir con una serie de reglas

Reglas básicas:

- Comunes a todos los documentos XML.
- Si no se cumplen se obtiene un error fatal.
- Si se cumplen el documento estará bien formado.

Reglas definidas por el usuario (DTD):

- Particulares de cada lenguaje XML.
- Si no se cumplen se produce un error de tipo documento inválido.
- Si un documento bien formado las cumple será un documento válido.

2.1.2 Características de XML

XML [2] es un conjunto de reglas para diseñar formatos de texto que permiten estructurar datos. XML no es un lenguaje de programación, pero sí permite que una máquina sea capaz de leer datos, generar datos y asegurarse que la estructura de los datos no es ambigua. Tiene algunas ventajas: es extensible, independiente de la plataforma, y soporta internacionalización y localización. Además es totalmente compatible con la codificación *Unicode* y tiene licencia libre.

2.1.3 Aplicaciones basadas en XML

A los lenguajes definidos por medio del metalenguaje XML se los denomina aplicaciones de XML. Un ejemplo de aplicación XML es RDF (Resource Description Framework). RDF es un formato de texto XML que permite la descripción de recursos, como las listas de reproducción de música, colecciones de fotos y bibliografías, por medio de metadatos. Por ejemplo, RDF te permite identificar a gente en un álbum web de fotos usando la información de una lista de contactos personal, así un cliente de correo electrónico podría automáticamente enviar un mensaje a la gente que aparece en determinadas fotos en la web. RDF provee herramientas para describir información con el objetivo de hacer la web un poco más semántica.

2.2 HTML 5

Para realizar este proyecto, se ha utilizado el estándar HTML5 (HyperText Markup Language, versión 5) [3]. El desarrollo de este lenguaje está regulado por el Consorcio W3C (World Wide Web Consortium).

HTML 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Sin embargo esta nueva versión no consiste sólo en esto, sino que es una nueva versión de diversas especificaciones, entre las que se encuentran:

- HTML 4
- XHTML 1
- DOM (Document Objetc Model) Nivel 2

Además HTML 5 pretende proporcionar una plataforma con la que desarrollar aplicaciones web más parecidas a las aplicaciones de escritorio, donde su ejecución dentro de un navegador no implique falta de recursos o facilidades para resolver las necesidades reales de los desarrolladores.

2.2.1 Novedades de HTML versión 5

HTML 5 incluye novedades significativas en diversos ámbitos. Como se comentó anteriormente, no sólo se trata de incorporar nuevas etiquetas o eliminar otras, sino que supone mejoras en áreas que hasta ahora quedaban fuera del lenguaje y para las que se necesitaba utilizar otras tecnologías.

- Estructura del cuerpo: La mayoría de las webs tienen un formato común, formado por elementos como cabecera, pie, navegadores, etc. HTML 5 permite agrupar todas estas partes de una web en nuevas etiquetas que representarán cada una de las partes típicas de una página.

- Etiquetas para contenido específico: Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones *Flash* o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.

- Canvas: Es un nuevo componente que permitirá dibujar, por medio de las funciones de una API (Application Programming Interface), en la página web todo tipo de formas, que podrán estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece *Flash*, pero dentro de la especificación del HTML y sin la necesidad de tener instalado ningún *plugin*.

- Bases de datos locales: El navegador permitirá el uso de una base de datos local, con la que se podrá trabajar en una página web por medio del cliente y a través de una API. Es algo así como las *cookies*, pero pensadas para almacenar grandes cantidades de información, lo que permitirá la creación de aplicaciones web que funcionen sin necesidad de estar conectados a Internet.

- *Web Workers*: Son procesos que requieren bastante tiempo de procesamiento por parte del navegador, pero que se podrán realizar en un segundo plano, para que el usuario no tenga que esperar a que se terminen para empezar a usar la página. Para ello se dispondrá también de una API para el trabajo con estos *Web Workers*.

- Aplicaciones *web offline*: Existirá otra API para el trabajo con aplicaciones web, que se podrán desarrollar de modo que funcionen también en local y sin estar conectados a Internet.

- Geo localización: Las páginas web se podrán localizar geográficamente por medio de una API que permita la Geo localización.

- Nuevas APIs para interfaz de usuario: Temas tan utilizados como el "*drag & drop*" (arrastrar y soltar) en las interfaces de usuario de los programas convencionales, serán incorporadas al HTML 5 por medio de una API.

- Fin de las etiquetas de presentación: Todas las etiquetas que tienen que ver con la presentación del documento, es decir, que modifican estilos de la página, serán eliminadas. La responsabilidad de definir el aspecto de una web correrá a cargo únicamente de CSS.

En el presente proyecto se ha utilizado el objeto *canvas*, para implementar el editor de diagramas en sí y se ha utilizado la API de '*drag & drop'* para la inserción de elementos en los diagramas. Es por ello por lo que en las próximas secciones describiremos con un poco más de detalle estos dos elementos.

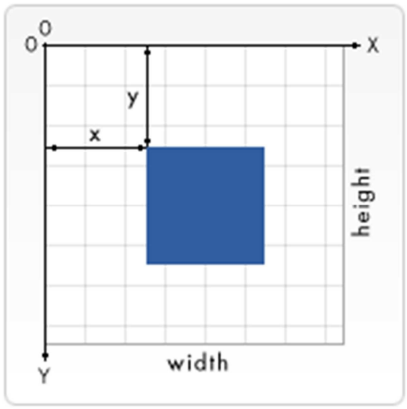
2.2.2 Canvas

Canvas [4] es un elemento de HTML versión 5, que se utiliza para dibujar gráficos desde *scripts* (normalmente *JavaScript*). Este elemento nos permite dibujar gráficos, hacer composiciones con fotos, juegos o realizar animaciones.

Fue creado inicialmente por *Apple* para el *Mac OS X Dashboard* y posteriormente se implementó en Safari. Los navegadores basados en *Gecko* 1.8 como *Firefox* a partir de la versión 1.5 también soportan este elemento, y está estandarizado por el WHATWG (Web Hypertext Application Technology Working Group)

El elemento *canvas* representa un lienzo en el que dibujar elementos, este posee dos atributos *width* (ancho) y *height* (alto) cuyo tamaño por defecto es de 300x150 píxeles.

A continuación se explicará en qué consiste el espacio de coordenadas del *canvas*. Normalmente una unidad corresponde con un píxel en el canvas. De esta manera, el origen de un *canvas* se establece en la esquina superior izquierda (coordenadas (0,0)). Todos los elementos se colocan con respecto a este origen, así la posición de la esquina superior izquierda del cuadrado azul estaría en la coordenada (x, y) respecto al origen. También es posible modificar las figuras, rotándolas y escalándolas respecto al *canvas*.



Por ejemplo podemos mostrar una descripción del contenido del *canvas* o introducir una imagen fija.

```
<canvas id="stockGraph" width="150" height="150">
  current stock price: $3.15 +0.15
</canvas>

<canvas id="clock" width="150" height="150">
  <img src="images/clock.png" width="150" height="150" alt=""/>
</canvas>
```

Canvas crea una superficie de dibujo de un tamaño fijo, éste puede tener un contexto de renderización, el siguiente código muestra la manera en que se obtiene una API de dibujo en 2D para el *canvas*.

```
var context = canvas.getContext('2d');
```

Mediante esta API es posible dibujar formas simples:

strokeRect`(x,y,width,height)`: Dibuja un rectángulo en las coordenadas pasadas.

lineTo`(x, y)`: Dibuja una línea en las coordenadas pasadas.

arc`(x, y, radio, startAngle, endAngle, sentidoantihorario)`: Dibuja un arco con centro en las coordenadas x e y, con un radio y grados de principio y fin pasados como argumentos, también se indicará el sentido del arco.

2.3 CSS (Cascading Style Sheets)

Las hojas de Estilo en Cascada [5], son un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos.

CSS se utiliza para dar estilo a documentos HTML y XML que se han comentado en capítulos anteriores, separando el contenido de la presentación. Los estilos definen la forma de mostrar los elementos HTML y XML. Cualquier cambio en el estilo marcado para un elemento en la hoja CSS afectará a todas las páginas que enlazan a esa CSS y en las que aparezca ese elemento.

2.3.1 CSS: Funcionamiento

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un *selector* y un bloque de declaraciones. A su vez cada una de las declaraciones del bloque está compuesta por una propiedad y el valor que se le asigne.

h1 {color: red;}

h1 es el selector
{color: red;} es la declaración

En el ejemplo anterior, el *selector* h1 indica que todos los elementos h1 se verán afectados por la declaración, donde se establece que la propiedad color va a tener el valor *red* (rojo) para todos los elementos h1 del documento o documentos que enlacen a esa hoja de estilos. Es posible definir cualquier propiedad del elemento: color, fuente, alineación del texto, tamaño y otras características no visuales como definir el volumen de un sintetizador de voz.

Es posible definir varios selectores separados por comas y varias declaraciones separadas por punto y coma. Normalmente se describe una propiedad por línea, de la siguiente manera:

```
h1 {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",Times, serif;
  color: red;
  background-color: #d8da3d;
}
```

Las tres formas más conocidas de dar estilo a un documento son las siguientes:

- Utilizando una hoja de estilo externa que estará vinculada a un documento a través del elemento <link>, el cual debe ir situado en la sección <head>.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>Título</title>
    <link rel="stylesheet" type="text/css" href="http://www.w3.org/css/officeFloats.css" />
  </head>
  <body>
  .
  .
  .
  .
  </body>
</html>
```

- Utilizando el elemento <style>, en el interior del documento al que se le quiere dar estilo, y que generalmente se situaría en la sección <head>. De esta forma los estilos serán reconocidos antes de que la página se cargue por completo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>hoja de estilo interna</title>
    <style type="text/css">

      body {
        padding-left: 11em;
        font-family: Georgia, "Times New Roman", serif;
        color: red;
        background-color: #d8da3d;
      }

      h1 {
        font-family: Helvetica, Geneva, Arial, sans-serif;
      }

    </style>
  </head>
  <body>
    <h1>Aquí se aplicará el estilo de letra para el Título</h1>
  </body>
</html>
```

Utilizando estilos directamente sobre aquellos elementos que lo permiten a través del atributo <style>. Hay que tener en cuenta que este tipo de definición del estilo pierde las ventajas que ofrecen las hojas de estilo al mezclarse el contenido con la presentación.

2.4 El lenguaje JavaScript

Toda la funcionalidad que se encuentra detrás de este proyecto está implementada en *JavaScript* [6].

JavaScript es un lenguaje de *scripting* compacto, diseñado específicamente para añadir interactividad a las páginas HTML. Es un lenguaje interpretado lo que significa que cada *script* se ejecuta directamente sin una compilación previa.

Los programas *JavaScript* van incrustados en los documentos HMTL, y se encargan de realizar acciones en el cliente, como puede ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos, etc.

El nombre oficial de *JavaScript* es ECMAScript ya que fue desarrollado y mantenido por la ECMA International cuyo estándar oficial es el ECMA-262. El lenguaje fue inventado por *Brendan Eich*, en la empresa *Netscape* y apareció en todos los navegadores de *Netscape* y *Microsoft* desde 1996. El estándar fue aprobado como un estándar internacional ISO (ISO/IEC 16262) en 1998, hoy en día su desarrollo sigue en proceso.

2.4.1 Cómo incluir código JavaScript

Para insertar un *JavaScript* en una página HTML se usa la etiqueta <script>. Dentro de esta etiqueta se usa el atributo *type* para definir el lenguaje de *scripting*. En el siguiente ejemplo las etiquetas <script type="text/javascript"> y </script> indican donde comienza y acaba el código *Javascript*.
El comando *document.write* es un comando estándar *JavaScript* para escribir texto en una página.

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

También es posible incluir codigo J*avascript* utilizando la etiqueta *script* con un atributo src.

```
<script type="text/javascript" src="[Ruta_archivo]"></script>
```

2.5 El Navegador Mozilla Firefox.

A continuación se explicarán las propiedades del navegador sobre el que trabajaremos, *Mozilla Firefox*.

Firefox es un navegador desarrollado por la *Mozilla Foundation* y un gran número de colaboradores que tienen como objetivo mantener la innovación en Internet.

Entre sus principales características destacan [7]:

- La personalización del propio navegador, mediante temas, *skins*, un gestor de complementos, etc.

- La privacidad permitiendo una navegación personal, borrando toda la información sobre las páginas visitadas.

- La seguridad mediante programas antivirus, control parental, bloqueador de ventanas emergentes y un gestor de contraseñas.

- Mejor rendimiento con protección ante fallos, un menor impacto de la memoria, con un recolector que limpia de forma continua la memoria no usada y una carga de páginas más veloz, gracias al motor de *JavaScript TraceMonkey*.

- Mejora la productividad mediante un lector de canales, un gestor de descargas, corrector ortográfico y la posibilidad de restaurar una sesión entre otros.

- Las búsquedas “inteligentes”, mediante palabras clave: al introducir un término en la barra de direcciones la función de autocompletado mostrará los sitios que se encuentren tanto en tu historial de búsqueda, como sitios que hayan sido marcados y etiquetados desde el desplegable.

- El acceso universal que permite una gran variedad de tipografías.

- La asistencia y ayuda al usuario, contando incluso con un *chat* en directo en el que los miembros de la comunidad resuelven dudas.

El código fuente de *Firefox* está disponible libremente bajo la triple licencia de *Mozilla* como un programa libre y de código abierto. La versión activa en el momento de escribir esta memoria es la 4.0.1 publicada el 28-04-2011, sin embargo en el contexto de este proyecto se ha utilizado la versión 3.6.13 que era la existente en el momento del desarrollo del mismo.

Firefox tiene además múltiples temas para personalizar su aspecto, complementos y extensiones que aumentan la funcionalidad de éste.

2.5.1 Drag & drop

Firefox y *Mozilla* soportan numerosas formas de realizar *drag and drop* [[8*]* .Esto permite al usuario hacer clic sobre un elemento, arrastrarlo a otra localización y soltar el botón del ratón para dejar el elemento en ese lugar. Mientras se traslada el elemento se puede ver una imagen traslucida del elemento que se está arrastrando. El destino puede ser otra aplicación diferente. Los sitios web, las extensiones y las aplicaciones XUL pueden hacer uso de esta funcionalidad para personalizar los elementos que se pueden arrastrar, etc.

Cuando se arrastra un elemento, hay que aportar distinta información:

- El objeto que será arrastrado, que puede ser texto, una imagen, etc.
- La imagen que aparecerá detrás del puntero mientras un elemento es arrastrado, esta imagen puede ser personalizada, sin embargo normalmente se genera una imagen por defecto en función del tipo de elemento a arrastrar.
- También se permiten algunos efectos de arrastre. Son posibles tres efectos:
    - copy → para indicar que el dato que está siendo arrastrado, se copiará de su localización actual al destino.
    - move →para indicar que el dato será movido.
    - link→ para indicar que se creara algún tipo de conexión entre el elemento origen y la localización destino.

El *drag & drop* se realiza a través de eventos, estos pueden ser:

- Dragstart: Se lanza cuando se empieza a arrastrar un elemento, durante este evento el escuchador necesita información sobre los datos que se arrastran y la imagen asociada.
- Dragenter: Se lanza cuando el ratón se mueve por primera vez sobre un elemento mientras un objeto está siendo arrastrado. El escuchador asociado indica si se permite soltar el objeto en esta localización o no.

- Dragover: Este evento se lanza mientras el ratón se mueve sobre un elemento mientras un objeto está siendo arrastrado.
- Dragleave: Este evento se lanza cuando el ratón deja un elemento mientras se arrastra un objeto. Los escuchadores deberían eliminar cualquier marca que indique que el objeto se puede soltar.
- Drag: Se lanza al principio del arrastre, esto es, el elemento donde se lanza el dragstar.
- Drop: Se lanza cuando se suelta el objeto al final de una operación de arrastre. El escuchador debería ser el responsable de obtener información sobre el objeto arrastrado y sobre el sitio donde se ha soltado.
- Dragend: Se lanza cuando se completa la operación de arrastre, sea esta exitosa o no.

2.6 XUL

XUL es un lenguaje basado en XML [9] para la descripción de interfaces de usuario que fue creado durante el desarrollo de *Firefox*.

En este proyecto se utiliza el lenguaje XUL que está basado en XML, metalenguaje que se ha comentado en la sección 2.1.

Aunque inicialmente se creó para facilitar y acelerar el desarrollo del navegador *Mozilla*, como está basado en XML y posee sus ventajas, actualmente permite crear potentes aplicaciones multiplataforma.

Gecko es el motor del navegador *Mozilla* y el encargado de interpretar las descripciones XUL. De hecho la interfaz completa del navegador *Mozilla* está implementada en ese lenguaje.

XUL consta de un amplio conjunto de componentes gráficos, entre otros:

- Controles de entrada como *TextBox, CheckBox*, etc.
- Barras de herramientas con botones y otros contenidos.
- Menús en una Barra de menú o diálogos emergentes (*pop-up*)
- Árboles
- *Overlay* (elemento adicional creado por XUL, en un *plugin* del navegador es de donde "se cuelga" lo que se le agrega a *Firefox*).

2.6.1 Tipos de Aplicaciones XUL

Hay cuatro tipos distintos de aplicaciones:

- Extensiones de *Firefox*

  Son barras de herramientas, menús u otros documentos XUL que agregan funcionalidades al navegador. Para hacer esto, se usa un elemento definido por XUL llamado "*overlay*", este elemento permite incorporar componentes o elementos de interfaz al propio navegador.

- Aplicaciones independientes

  Estas aplicaciones son creadas mediante *XULRunner*, el cual es una versión de la plataforma *Mozilla* que permite crear aplicaciones XUL independientes. No es necesario el navegador para ejecutar estas aplicaciones ya que tienen su propio ejecutable.

- Paquete XUL

  Es un punto intermedio entre los dos tipos anteriores, es instalado como una extensión, pero actúa como una aplicación separada del navegador. Este tipo de aplicaciones son creadas cuando no queremos utilizar una aplicación *XULRunner* completa.

- Aplicaciones XUL Remotas

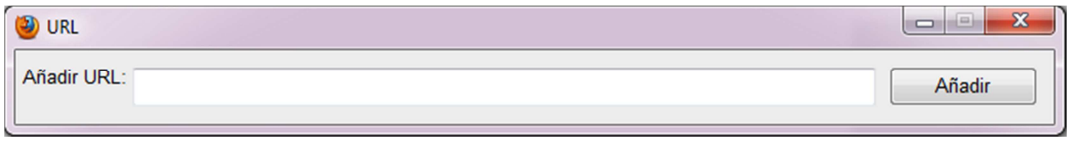
  Son aplicaciones XUL que se encuentran en un Servidor *Web* y son ejecutadas remotamente como cualquier otra página *Web*.

Como ya se ha comentado anteriormente, es muy sencillo implementar un interfaz de usuario utilizando XUL, pondremos como ejemplo una porción de código de este proyecto:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="chrome://editor/skin/administrador.css" type="text/css"?>
<!DOCTYPE window SYSTEM "chrome://editor/locale/url.dtd">

<window id="Ventana url"
title="&titulo;"
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
class="ventana"
onload="cargar();">
 <script type="text/javascript" src="url.js"></script>
 <script type="text/javascript" src="utilidades.js"></script>
 <script type="text/javascript" src="io.js"></script>
    <vbox>
      <hbox>
        <label id="etiqueta" control="urls" value="&anadir.url;" />
        <textbox  id="urls" multiline="false" width="500px" />
        <button id="anadir"  class="diagrama" label="&añadir;" onclick="anadirUrl();" width="100px"> </button>
      </hbox>
    </vbox>
</window>
```

El código corresponde a la siguiente pantalla:



Como es un fichero XML comienza con la declaración propia de estos archivos, así como la referencia a los CSS y DTDs correspondientes.

Posteriormente están las etiquetas que permiten cargar todos los *scripts* que utilizaremos, en nuestro caso son tres *JavaScript*.

Luego vemos que el elemento principal de XUL es una ventana, donde especificamos el título, y la función que se ejecutará al cargar la ventana.

La manera principal de organizar los distintos elementos de la interfaz es el 'Modelo de cajas'. Este modelo permite dividir una ventana en cajas, cuyos elementos se orientarán de forma horizontal o vertical, además existen distintos elementos espaciadores que permiten controlar el diseño de la ventana.

En nuestro ejemplo vemos que la ventana principal se organiza en una caja con orientación horizontal (<*hbox*>) dentro de otra con orientación vertical (<*vbox*>).

Dentro se encuentran tres elementos diferentes:

Un <*label*> donde introducimos un texto descriptivo a mostrar, usaremos la propiedad *control* para asociar la etiqueta al *textbox*.

Un <*textbox*> con una anchura de 500 píxeles y con la propiedad *multiline* a *false* para indicar que sólo se muestre una línea.

Y finalmente un <*button*> en el que indicamos el método a ejecutar, el tamaño y el texto.
Es posible personalizar los botones con imágenes en lugar de texto, mediante la propiedad *image*.

2.7 Desarrollo de una extensión Firefox.

En esta sección describiremos brevemente los elementos de los que consta típicamente una extensión *Firefox* [10], y que habrá que desarrollar para implementar la extensión objetivo de este proyecto. Las extensiones *Firefox* se distribuyen en un fichero .xpi que tiene la siguiente estructura:

```
extension.xpi:
                /install.rdf
                /components/*
                /components/cmdline.js
                /defaults/
                /defaults/preferences/*.js
                /plugins/*
                /chrome.manifest
                /chrome/icons/default/*
                /chrome/
                /chrome/content/
                /chrome/skin/
                /chrome/locale/
```

En esta estructura destacan dos ficheros importantes: el fichero install.rdf y el fichero chrome.manifest que describimos en las subsecciones (2.6.1) y (2.6.3) respectivamente.

Además, para entender mejor la función del fichero chrome.manifest, explicaremos un poco en qué consisten los *chromes* en la subsección (2.6.2).

2.7.1 El archivo install

Este fichero proporciona una descripción de los aspectos de instalación de la extensión y contiene lo siguiente:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <em:id>chifled@mydomain.com</em:id>
    <em:version>1.0</em:version>
    <em:type>2</em:type>

    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>3.5</em:minVersion>
        <em:maxVersion>3.6.*</em:maxVersion>
      </Description>
    </em:targetApplication>

    <em:name>Chickenfoot Flowchart Editor</em:name>
    <em:description>Extensión para la edición gráfica de aplicaciones basadas en Chickenfoot </em:description>
    <em:creator>Beatriz Lopez Morenc</em:creator>
  </Description>
</RDF>
```

A continuación se detallan las propiedades más relevantes:

- **Id** - El identificador de la extensión que será un GUID o una cadena con formato del tipo nombreextension@organizacion.tld.

- **Versión** - La cadena de la versión actual del complemento.

- **Type -** Valor de tipo entero que representa el tipo de complemento, en nuestro caso este valor será 2 que es el que representa las extensiones. Este campo puede tener otros valores como el 4 para el caso de los Temas, el 16 para los *plugin* o el 32 para los paquetes.

- **TargetApplication -** Especifica la aplicación a la que se refiere este complemento, los elementos *minVersion* y *maxVersion* permiten al desarrollador de la extensión especificar con qué versiones de *Firefox* han sido probadas.

- **Name** - El nombre del complemento que aparecerá al instalarlo.

- **Description** - La descripción del complemento.

- **Creator -** El nombre del desarrollador principal.

2.7.2 Paquetes chrome y Chrome URIs (Uniform Resource Identifier)

Los archivos XUL forman parte de *Chrome Packages* - paquetes de componentes de interfaz del usuario, los cuales se cargan a través de la URI con prefijo chrome://.

Más que cargar el navegador y sus extensiones desde el disco utilizando un archivo (file://), dado que la ubicación de *Firefox* en un sistema puede cambiar de una plataforma a otra y de un sistema a otro, los desarrolladores de *Mozilla* se decantaron por una solución de cargar componentes en base a un identificador creado respecto al contexto de XUL. Así por ejemplo, la ventana del navegador es: chrome://browser/content/browser.xul

Las URIs chrome constan de varias partes diferenciadas:

- En primer lugar, la *URI scheme* (esquema URI) informa a la librería de red de *Firefox* de que es un *'Chrome URI'* y que el contenido que se cargue debe ser manejado de manera especial.

- En segundo lugar, un nombre de paquete (en nuestro caso utilizamos **editor**), que identifica al componente en la interfaz del usuario. Este nombre debe ser único, de esa manera se evitará el conflicto entre componentes (navegador y/o extensiones).

- En tercer lugar, el tipo de información que ofrece el archivo. Hay tres tipos:

    - *content* XUL, *JavaScript*, XBLs (**X**ML ***B****inding* ***L****anguage*), etc, que forman la estructura y el comportamiento de una aplicación.

    - *locale* (DTD, archivos de propiedades, etc, que contienen cadenas para la localización de la interfaz de aplicación). Todo el texto que se muestra en las ventanas se almacena de forma separada, así resulta más sencillo traducir la extensión a distintos idiomas.

    - *skin* (CSS (***C****ascading* ***S****tyle* ***S****heets*) e imágenes, las cuales definen la apariencia de la interfaz. Se almacenan de forma separada a los archivos XUL para facilitar modificar el aspecto de una aplicación.

- En último lugar, la ruta del archivo a cargar.

Por lo tanto, chrome://editor/content/images/flecha.jpg carga el archivo flecha.jpg de la sección images, del *content* de editor.

2.7.3 El fichero chrome.manifest

Este fichero debe contener lo siguiente:

```
content     editor    content/
overlay  chrome://browser/content/browser.xul chrome://editor/content/overlay.xul
locale  editor  es-ES  locale/es-ES/
```

En la primera línea se especifica, primero el tipo de material dentro de un paquete *chrome* seguido del nombre del paquete (editor) y en tercer lugar la localización de los archivos del paquete *chrome*.

Esta línea dice que debemos encontrar los archivos de *content* en la ruta content/, la cual es relativa a la ruta de chrome.manifest.

La segunda línea le dice a *Firefox* que fusione overlay.xul con browser.xul cuando browser.xul se cargue. En este caso, la extensión se fusionará con browser.xul que es el navegador en sí ya que ésta aparecerá en la barra de Herramientas de *Firefox*.

La tercera línea específica el *locale* que se aplica y el paquete al que se le aplica.

2.8 Chickenfoot

Chickenfoot [11] es una extensión de *Firefox* que añade un entorno de programación al navegador para escribir *scripts* que permitan la manipulación de las páginas *web* y automaticen la navegación *web*. En *Chickenfoot* los *scripts* se escriben en un conjunto de *JavaScript* que incluye funciones específicas para realizar estas tareas.

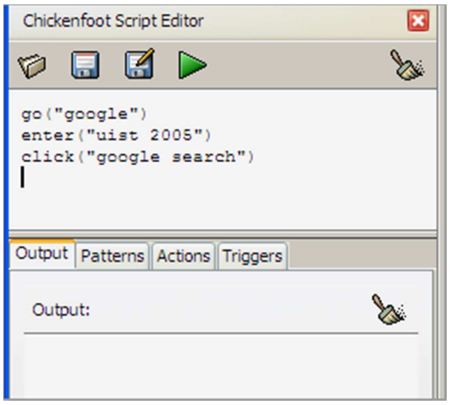
Escribiendo *scripts* simples en *Chickenfoot*, es posible alterar el comportamiento de los sitios *web* y hacer que las paginas respondan de diferente manera, personalizándolas. Los *scripts* se pueden crear usando solo el navegador (mediante una extensión descargable) y algún código simple.

El funcionamiento de *Chickenfoot* se basa en la manipulación del DOM del navegador. La posibilidad de manipular el DOM permite cambiar el comportamiento del código en una *web*, permitiendo la interacción con la página.

Se pueden crear fácilmente aplicaciones que permitan cambiar el color de las páginas, añadir fotos, rellenar formularios automáticamente, navegación automática por una *web*, cambiar el tamaño de los cuadros de texto, encontrar el número de enlaces en una página etc.

Los *scripts* se pueden guardar y cargar, lo que permite escribir código mientras se navega y después lanzarlo en cualquier momento.

La siguiente imagen muestra la extensión *Chickenfoot* con un *script* que abre la web de *Google*, introduce el texto “uist 2005” y hace clic sobre el botón o enlace que contenga el texto “google search”.



2.8.1 Características de Chickenfoot

- Navegación Simple.

  La navegación simple es probablemente la principal característica de *Chickenfoot*, por ejemplo para abrir una página *web*, rellenar un cuestionario y pulsar un botón el código sería el siguiente:

```
go("webmonkey.com")
enter("search","chickenfoot")
click("second button")
```

- Comparación de patrones.

  Chickenfoot incluye una comparación de patrones que permite por ejemplo conocer el número de veces que aparece la palabra "*chickenfoot*" en una página *web*.

```
m=find("chickenfoot")
m.count
```

- Modificación de Páginas.

  Chickenfoot permite también cambiar el código al vuelo. Por ejemplo, si queremos modificar una imagen en una página, como puede ser reemplazar un *logo* con una imagen propia, entonces el código sería el siguiente.

```
replace("http:www.webmonkey.com/logo.gif", "http://myserver/mylogo.gif")
```

Este proyecto consiste en una extensión para *Firefox* que permita la edición gráfica mediante diagramas de flujo y la posterior ejecución de código *Chickenfoot* para realizar aplicaciones sencillas que permitan modificar la *web* añadiéndole funcionalidad. La herramienta será validada mediante la implementación de *scripts* que permitan automatizar ciertas tareas en *Gmail*.

Capítulo 3

Requisitos de Diseño

Al comienzo de este proyecto, se plantearon una serie de requisitos de diseño a cumplir por parte de la aplicación desarrollada, los cuales se enumerarán a continuación:

1. El presente proyecto deberá consistir en una extensión para el navegador *Mozilla Firefox* que permita a un desarrollador, realizar aplicaciones basadas en *Chickenfoot* de forma gráfica, mediante diagramas de flujo.

2. La interfaz gráfica se debe implementar en XUL mientras que la funcionalidad se implementará en *JavaScript*.

3. La extensión deberá comprobar al inicio de su ejecución si está instalado el complemento de *Chickenfoot* y, en caso contrario, se mostrará un mensaje indicando la URL desde la cual se podrá descargar.

4. La extensión constará de un administrador de diagramas, donde se podrán crear, modificar, borrar, importar o exportar y asociarles URLs.

5. Además se implementará un editor gráfico de diagramas, el cual deberá consistir en una ventana con una barra de herramientas y un panel central, donde el desarrollador podrá arrastrar y soltar los distintos elementos que compondrán el diagrama.

6. Antes de abrir el editor se deberá solicitar un identificador para el diagrama que se va a crear, no se continuará hasta que el usuario no haya introducido un identificador válido y único en la aplicación.

7. El panel central del editor deberá utilizar el elemento *canvas* de HTML 5.0.

8. El título de la ventana que contendrá el editor, mostrará el nombre del diagrama que se esté creando o modificando en ese momento.

9. Las herramientas de las que constará el editor serán las siguientes:

    - Una caja que representará un proceso y que contendrá el código *Chickenfoot*.
    - Una flecha para unir los distintos procesos y que representará el flujo de datos de entrada y salida entre estos.
    - Una caja doble que podrá contener otro diagrama, de manera que se puedan crear diagramas más complejos (anidados).
    - Una fuente de datos donde se indicará una URL de la cual se podrán obtener sus datos para utilizarlos dentro del diagrama.
    - Un elemento que represente el final del diagrama.

10. Para añadir código a un proceso se deberá hacer doble clic en una caja, además el usuario podrá elegir entre cualquier editor de texto que tenga instalado en el PC, o mostrar el código sobre una nueva ventana, siendo esta opción configurable por el usuario.

11. Se permitirá abrir varias cajas de edición de código a la vez.

12. Un proceso sólo podrá tener una salida.

13. Toda la información de configuración del editor deberá estar almacenada en un fichero.

14. Se visualizarán los elementos mientras se mueven en la pantalla, además será posible borrar los elementos uno a uno o todo el diagrama a la vez.

15. Al realizar clic sobre un elemento del diagrama, dicho elemento quedará seleccionado.

16. Todos los elementos deberán tener un identificador único dentro del diagrama, al usuario se le deberá ofrecer uno por defecto y este será visible en el *canvas*.

17. Además al situar el ratón encima de la fuente de datos, se deberá mostrar en una ventana emergente la URL que representa.

18. Los identificadores de los distintos elementos se podrán modificar en cualquier momento.

19. Las flechas indicarán los datos de salida y entrada entre procesos, los identificadores de estos elementos se podrán utilizar dentro de los procesos representando el flujo de los datos de entrada y salida.

20. Las aplicaciones finales creadas por el desarrollador, se deberán ejecutar cuando se introduzca en el navegador una URL determinada.

21. Al hacer clic dos veces sobre una caja compleja se deberán mostrar al usuario las opciones de ver el código final del diagrama que contiene, o abrir un nuevo editor con ese diagrama.

22. Al guardar un diagrama, se intentará crear el *script* final que se ejecutará, si el diagrama no está completo o bien formado se le mostrará un mensaje al usuario indicándole que no se ha generado correctamente el código.

23. El usuario final solo verá la relación entre una URL y el script final que se ejecutará.

24. Se incorporará un mecanismo de bitácora (*log*) de información del sistema, de forma que se puedan imprimir mensajes de error de la aplicación.

Capítulo 4

Diseño de alto nivel

En este capítulo se expondrán las diferentes situaciones en las que se puede encontrar el usuario durante el manejo de esta aplicación. Se mostrarán las distintas pantallas y opciones desde un alto nivel de abstracción.

4.1 Acceso a la aplicación desde el navegador.

El primer paso para acceder a la aplicación después de su instalación a partir del paquete editor.xpi consistirá en abrir el navegador.

Además también será necesario instalar la extensión de *Chickenfoot* para el correcto funcionamiento, como se ha comentado en el capítulo anterior, en el caso de que la aplicación detecte que no está instalada, mostrará una ventana informativa con la dirección desde la cual podemos descargarla.

Una vez abierto el navegador podremos acceder a la extensión desde el menú Herramientas haciendo clic en la última opción: *ChickenfootFlowChartEditor*.

4.2 Descripción de la extensión.

La presente aplicación deberá constar de dos submenús, el administrador de diagramas, y el editor de estos.

- Administrador de diagramas

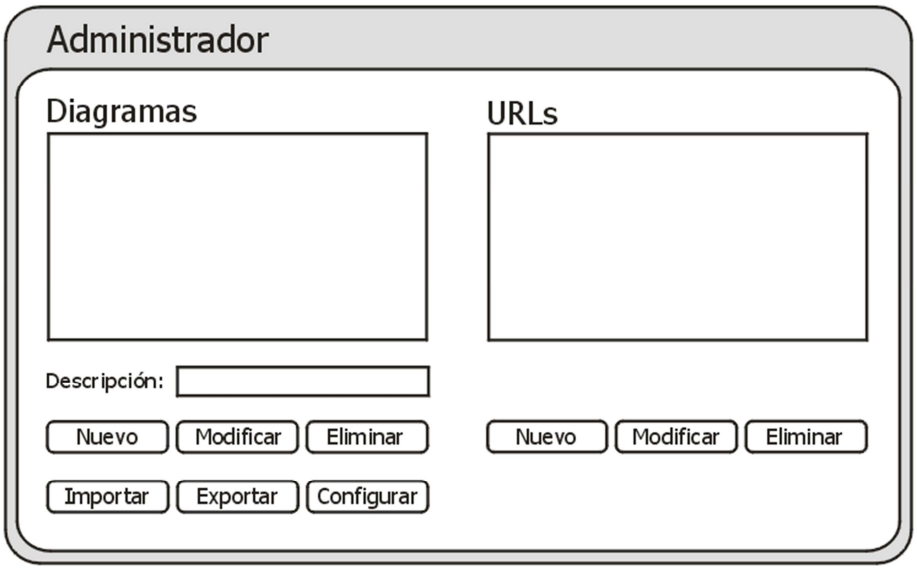
  La aplicación debera tener un administrador donde el desarrollador de aplicaciones en *Chickenfoot,* sea capaz de realizar acciones sobre los distintos diagramas creados.

  La funcionalidad de la que constará el administrador será:

  - Edición de un diagrama ya creado.
  - Eliminación por completo de un diagrama.

- Se debe permitir la importación y exportación de diagramas para permitir su uso en máquinas distintas.
- Asociación de una URL, a partir de la cuál, al insertarla en el navegador se ejecutará el *script* correspondiente al diagrama seleccionado.
- Modificación y eliminación de URLs asociadas a los diagramas.

La vista del administrador será una pantalla similar a la siguiente:



La primera vez todos los campos aparecerán vacíos, pero posteriormente mostrarán información sobre los distintos diagramas almacenados en el sistema.

Así al seleccionar cada diagrama en la parte izquierda se deberá mostrar su descripción y las URLs que tendrá asociadas.

- Editor de Diagramas

La aplicación deberá tener un editor, el cual permitirá al desarrollador de aplicaciones en *Chickenfoot*, generar *scripts* finales mediante una interfaz gráfica que le permita el diseño de diagramas de flujo.

Cada vez que se acceda al editor, la aplicación solicitará un nombre para identificar el diagrama que se va a crear, este identificador deberá ser único y no se podrá modificar posteriormente.

Una vez introducido, se abrirá la pantalla del editor sobre la que el usuario podrá comenzar a crear un diagrama de flujo.

Esta ventana tendrá varias partes:

En la parte superior se mostrará el nombre del diagrama que se está editando.

La parte central contendrá un panel en blanco sobre el que se irá formando el diagrama de flujo, este panel deberá utilizar el elemento *canvas* de HTML 5 que se ha nombrado anteriormente en el capítulo 2.

A la izquierda de la ventana se presentará la barra de herramientas con todos los elementos disponibles para formar un diagrama.

Desde esta barra se irá arrastrando cada elemento hasta el *canvas*.

Los elementos disponibles serán:

- Caja de código compleja: Podrá contener otras cajas sencillas o complejas en su interior, permitiendo así ampliar la complejidad de los *scripts* finales.

- Caja de código sencilla: Representará un proceso en el diagrama y contendrá el código asociado.

- Fuente de datos: Contendrá una URL de la cual se podrán cargar sus datos para utilizarlos dentro del diagrama.

- Punto final del diagrama: Todo elemento deberá tener un único punto final o de cierre del diagrama.

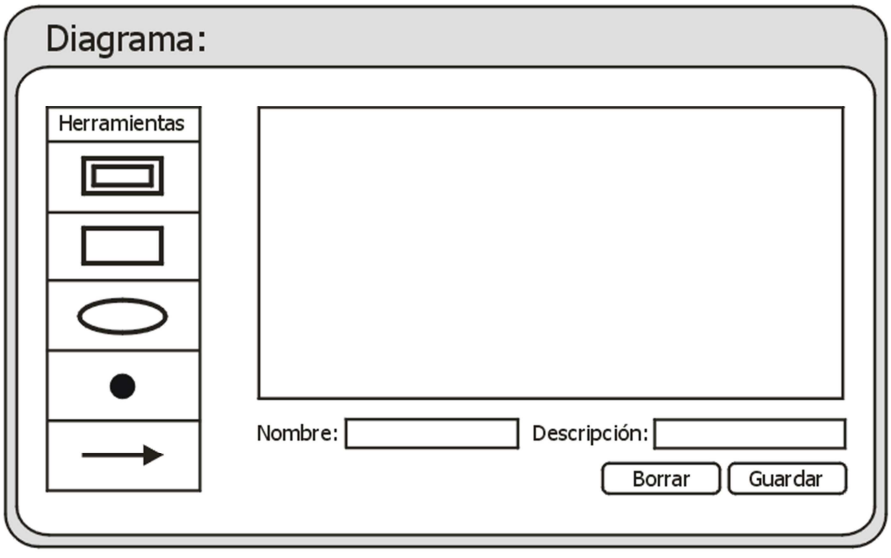
- Flecha de unión: Representa el flujo de datos entre procesos.

Además en la parte inferior deberá aparecer el nombre del diagrama que se está editando, su descripción, que se podrá modificar desde esta ventana, y varios botones:

- Borrar: Eliminará el diagrama creado.

- Guardar: Guardará el diagrama en el sistema, se deberán llevar a cabo varias comprobaciones tales como que el diagrama tenga un punto final y que todos los procesos contengan código, en caso contrario se guardará el diagrama pero se deberá avisará al usuario de que no se ha generado el *script* final.

La vista del editor será una pantalla similar a la siguiente:



4.3 Creando un diagrama.

El primer paso para crear un diagrama será insertar los elementos que formarán este.

Una vez que el usuario arrastre un elemento al *canvas*, la aplicación le solicitará un identificador, esta ventana deberá mostrar uno por defecto el cual se podrá modificar siempre y cuando sea único dentro del diagrama. En el caso de las fuentes de datos además se solicitará la URLs sobre la que se desea obtener los datos.

El siguiente paso será escribir el código. Para introducir el código en una caja bastará con hacer doble clic sobre esta.

La primera vez que se inserte código en un diagrama deberá aparecer una ventana de selección para elegir el ejecutable del editor de texto que tengamos instalado en la máquina y que queramos utilizar para escribir el código. Si se pulsa sobre cancelar, la aplicación abrirá un *textarea* donde podremos escribir nuestro código. El editor elegido se podrá modificar en cualquier momento mediante el botón Configurar del administrador.

Una vez tengamos los procesos con sus códigos deberemos unirlos para formar el diagrama correctamente, para ello se utilizarán las flechas.

Bastará con seleccionarlas en el panel de herramientas y hacer clic sobre los elementos que queremos unir, se deberá introducir también un identificador que se corresponderá con los datos de entrada del segundo proceso, así se podrá hacer referencia a estos dentro del código. Como se ha comentado anteriormente en el capítulo 3, la salida de cada proceso deberá ser única.

Una vez tengamos el diagrama definido, lo guardaremos mediante el botón correspondiente, entonces la aplicación comprobará que tenemos un diagrama con todos los elementos conectados entre sí correctamente y que tenga un punto final, si no es así se mostrará una ventana de error.

Cuando se hayan seguido todos estos pasos, la aplicación generará un único *script* final que será el que se ejecute al introducir la URL asignada al diagrama correspondiente.

Capítulo 5

Implementación

En este capítulo se va a describir con mayor detalle cuál ha sido la implementación llevada a cabo para el desarrollo de la extensión *ChickenfootFlowChartEditor*.

En concreto, a la hora de describir la implementación llevada a cabo se describirán por un lado aspectos relacionados con la interfaz gráfica de usuario, mientras que por otro se detallarán los ficheros de instalación de la extensión, la funcionalidad que contienen los distintos *scripts* utilizados para la lógica de la aplicación, y los ficheros de configuración generados por la propia extensión para el manejo de los diagramas.

5.1 La extensión ChickenfootFlowChartEditor

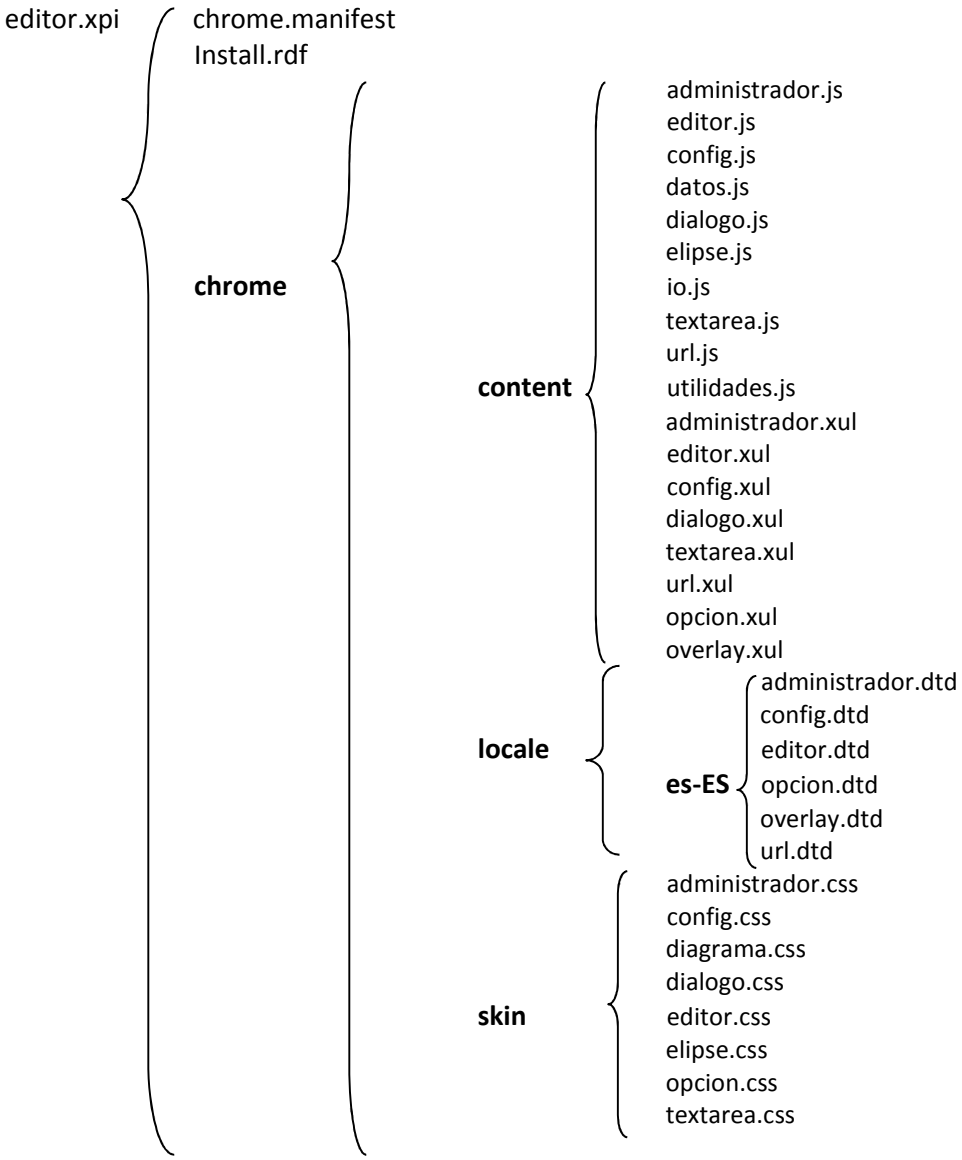
Como ya se ha comentado anteriormente, las extensiones para *Firefox* son pequeñas aplicaciones que ofrecen al navegador una funcionalidad extra.

En este apartado describiremos cómo se ha desarrollado esta extensión, cumpliendo con los requisitos expuestos en el capítulo 3 y siguiendo los aspectos de diseño recogidos en el capítulo 4.

5.1.1 Estructura de la extensión

Como se ha comentado anteriormente en el capítulo 2, el paquete en el que se encuentra la extensión implementada será un archivo XPI. Este paquete contiene los ficheros necesarios para el funcionamiento de esta. El contenido no es arbitrario sino que debe tener una estructura básica que seguirán todas las extensiones para *Firefox*. Este paquete permite instalar la extensión de una manera sencilla y también su distribución por *Internet*.

El paquete de esta extensión se denomina editor.xpi y tiene la siguiente estructura:



El fichero XPI no es más que un *zip*, así que es posible descomprimir el paquete para acceder a sus ficheros.

Como podemos ver en el diagrama anterior, en el primer nivel se encuentran dos archivos de configuración: el *chrome.manifest* y el *install.rdf*, así como un directorio *chrome* con tres directorios más que contendrán la funcionalidad de la extensión: *content*, para los *JavaScript* y ficheros XUL del interfaz de usuario, *locale* para los DTD y *skin* donde estarán los CSS.

En los próximos capítulos se explicarán las distintas partes del paquete XPI.

5.1.2 Fichero install.rdf

Como se ha comentado en el capítulo 2, el fichero *install.rdf* es el manifiesto de instalación que utiliza la aplicación XUL para determinar la información sobre cómo debe ser instalada una extensión determinada. Este archivo contiene metadatos que identifican la extensión, aportando información sobre su autor, sobre cómo se puede obtener más información, la versión de *Firefox* con las que funcionará correctamente, cómo se puede actualizar y otras.

El formato del manifiesto es RDF/XML.

El archivo debe ser llamado `install.rdf` y ubicarse en el nivel superior del paquete XPI de una extensión.

El esquema principal de este fichero se muestra a continuación:

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <!-- properties -->
  </Description>
</RDF>
```

Algunas de estas propiedades son obligatorias mientras que otras son opcionales.

Las propiedades obligatorias para que la aplicación se instale correctamente son: *id, version, type, targetApplication* y *name*.

A continuación veremos, el fichero install.rdf de la extensión desarrollada en el presente PFC.

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:em="http://www.mozilla.org/2004/em-rdf#">
 <Description about="urn:mozilla:install-manifest">
   <em:id>chifled@mydomain.com</em:id>
   <em:version>1.0</em:version>
   <em:type>2</em:type>

   <em:targetApplication>
     <Description>
       <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
       <em:minVersion>3.5</em:minVersion>
       <em:maxVersion>3.6.*</em:maxVersion>
     </Description>
   </em:targetApplication>

   <em:name>Chickenfoot Flowchart Editor</em:name>
   <em:description>Extensión para la edición gráfica de aplicaciones basadas en Chickenfoot </em:description>
   <em:creator>Beatriz Lopez Morenc</em:creator>
 </Description>
</RDF>
```

Como podemos ver, en este fichero se especifican metadatos como el identificador de la extensión, el nombre, su autor, la versión y una breve descripción.

Ya que este fichero de instalación es común para todos los complementos de *Mozilla*, será necesario especificar mediante la etiqueta *targetApplication*, la aplicación objetivo de la extensión, en este caso en el elemento <em:id> se ha especificado el GUID de *Firefox*.
Además se garantiza su correcto funcionamiento entre las mínimas y máximas versiones especificadas mediante las etiquetas <em:minVersion> y <em:maxVersion>.
Estas cadenas de versión son formateadas del mismo modo que la propiedad versión y será comparada con la versión de la aplicación, permitiendo que el autor de la extensión especifique sobre qué versiones de *Firefox* funcionará esta.

5.1.3 Fichero chrome.manifest

El motor de renderizado de *Firefox* denominado *Gecko*, mantiene un servicio que registra los distintos *chrome*. Este servicio provee el mapeo entre los nombres de los paquetes y las localizaciones físicas en disco de estos paquetes.

Para informar a este registro sobre los *chrome* disponibles, se usa un archivo de manifesto llamado *chrome.manifest*.

A continuación mostramos el manifiesto usado en esta extensión y describiremos su contenido:

```
content     editor    content/
overlay chrome://browser/content/browser.xul
chrome://editor/content/overlay.xul
locale  editor  es-ES  locale/es-ES/
skin        editor    skin/
```

A continuación se describirá qué hace cada línea de texto:

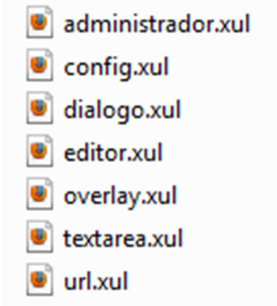
- **Línea 1:** mapea el contenido de *chrome://editor/content* a la carpeta *content*.
- **Líneas 2 y 3**: registran un *overlay* de la extensión para indicar a *Firefox* que debe mezclar la ventana principal del navegador (*chrome://browser/content/browser.xul*), con el fichero .xul indicado (*chrome://editor/content/overlay.xul*).
- **Línea 4:** registra un paquete *locale*, el nombre del *locale* es un identificador del idioma, o del idioma y el país como en este caso (es-ES) y se utilizará en el caso de que se instale más de uno, proporcionando el más apropiado en función de las preferencias del usuario.
- **Línea 5:** mapea el contenido de *chrome://editor/skin* a la carpeta *skin*.

5.1.4 Contenido (content)

Este directorio contiene todos los ficheros XUL que describen la interfaz gráfica del usuario y los archivos *JavaScript* que se encargan del funcionamiento de la aplicación.

A continuación se detallarán los distintos ficheros utilizados.

Vista (interfaz de usuario):

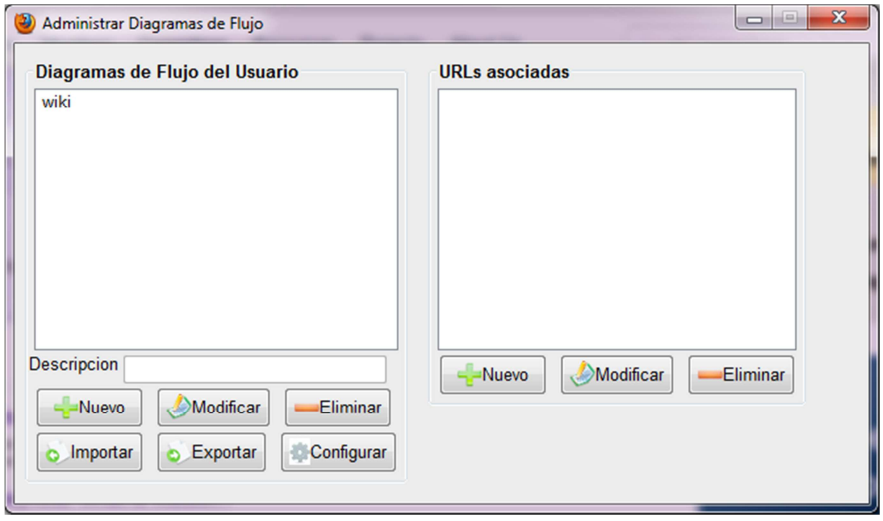


- overlay.xul

  Este fichero es el punto de unión con el navegador, como ya se ha comentado anteriormente, define el *overlay* y se integrará al final del menú de Herramientas del navegador.

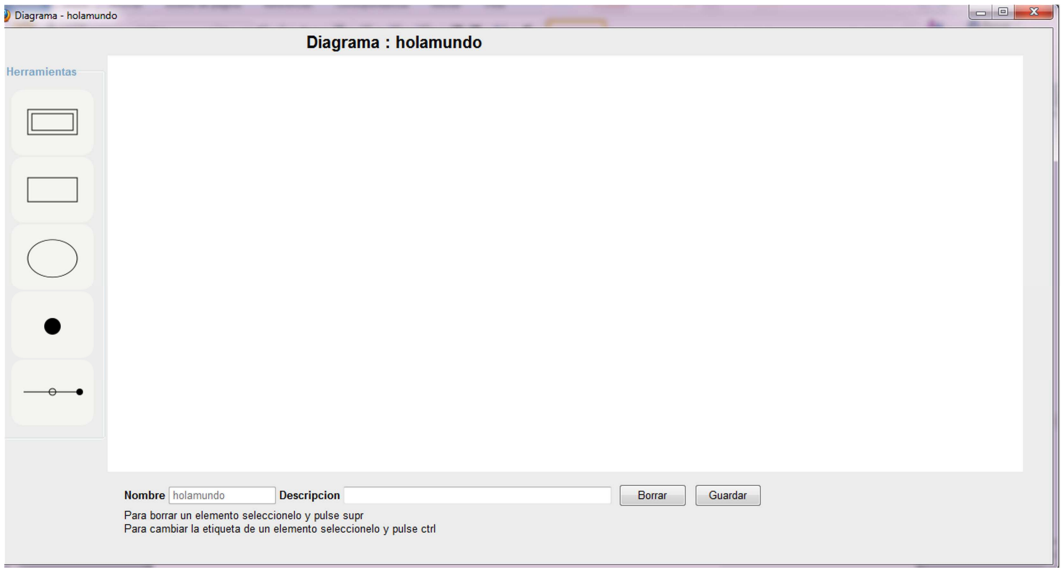
- administrador.xul

  Este fichero define la pantalla principal de la aplicación a través de la cual podemos administrar los distintos diagramas y las URLs asociadas.



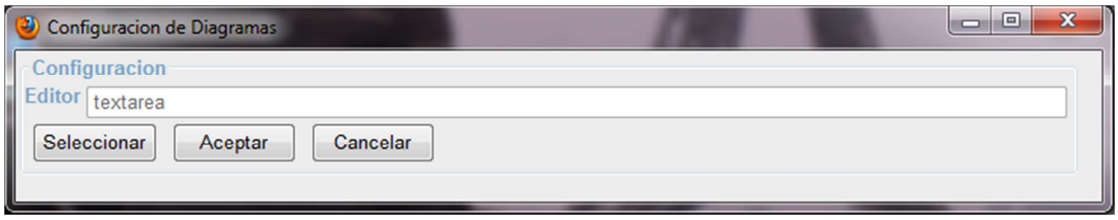
- editor.xul

  Es la ventana principal del editor de diagramas, a la izquierda tiene una barra de herramientas con todos los elementos para crear estos, en el centro está el canvas sobre el que se crea el diagrama y en la parte inferior se sitúan la descripción y los botones que permiten guardar, borrar o configurar el editor que se mostrará para modificar el código *Chickenfoot*.



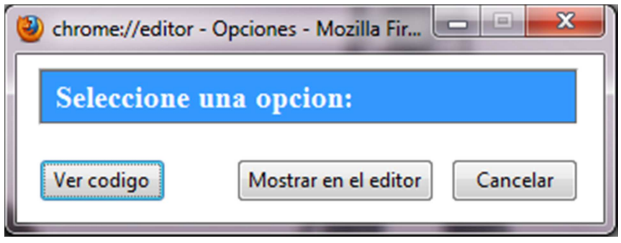
- config.xul

  Define la ventana de configuración donde se selecciona el editor que se abrirá para escribir el código *Chickenfoot*.



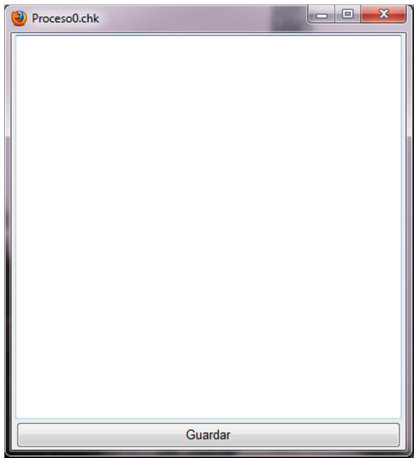
- dialogo.xul

  Es la pantalla que permite elegir, al hacer doble clic sobre una caja compuesta, entre mostrar el código final del diagrama contenido o ver el diagrama en un editor.



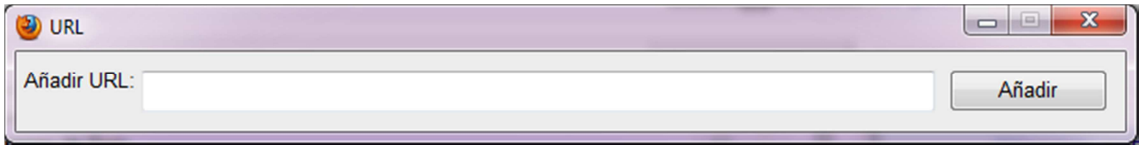
- textarea.xul

  Define la ventana sobre la que se escribe el código *Chickenfoot* en caso de no elegir ningún editor. Muestra un espacio donde escribir el código y un botón para guardarlo almacenándolo en el directorio propio de la extensión (dentro del perfil de *Firefox*), en una carpeta denominada igual que el diagrama y cuyo nombre será el que se haya elegido para el proceso que estemos editando, con extensión .chk.

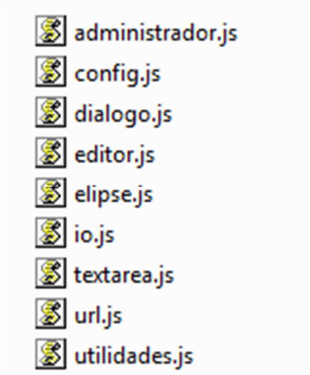


- url.xul

  Este fichero define la ventana en la que se escribirá la URL asociada a un diagrama.



Lógica de aplicación:



- administrador.js

  Es el fichero que contiene el código encargado de las acciones de la ventana de administración. En él se definen las funciones para cargar los datos de los diagramas instalados y mostrar así su información y las URLs asociadas. Engloba toda la funcionalidad de la administración de los distintos diagramas, con funciones para abrir el editor pasándole los datos del diagrama seleccionado, añadir, modificar o eliminar URLs asociadas así como para importar o exportar otros diagrama al directorio seleccionado.

- config.js

  Contiene el código encargado de modificar o añadir a la extensión el editor de texto que se usará para editar los *scripts Chickenfoot*.

- dialogo.js

  Contiene la funcionalidad para abrir los diagramas complejos en una ventana del editor nueva y de abrir el código del diagrama complejo seleccionado en el editor que ha elegido el usuario anteriormente. Si no se ha seleccionado ninguno se mostrará el código en un *textarea*.

- editor.js

  Contiene el código encargado de llevar a cabo todas las operaciones del editor, en su interior se definen las funciones para representar gráficamente los elementos del diagrama en el *canvas*. Además incluye la funcionalidad para añadir y generar el script

final, para guardar en un fichero XML todos los datos del diagrama y para cargar los mismos en el caso de que se abra un diagrama ya creado.

- elipse.js

  Contiene el código encargado de recoger y almacenar la información proporcionada al insertar en el diagrama una entrada de datos.

- io.js

  Define las funciones básicas de lectura y escritura de directorios y ficheros en disco, en función del sistema operativo en el que se encuentre instalado el navegador.

- textarea.js

  Contiene el código encargado de leer y escribir el código introducido en el *textarea* en el fichero correspondiente.

- url.js

  Contiene el código encargado de añadir, modificar y eliminar las URLS asociadas a un diagrama en el fichero de configuración de este.

- utilidades.js

  Contiene el código *JavaScript* encargado de ejecutar el código *Chickenfoot* correspondiente cuando se introduce una URL en el navegador. Además contiene funciones para escribir y leer los directorios y ficheros concretos, necesarios para el funcionamiento de la extensión, utiliza las funciones definidas en el fichero io.js.

5.1.5 Aspecto (skin)

En este directorio se encuentran todos los ficheros CSS encargados de definir la presentación de la interfaz.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación, permitiendo modificar su apariencia editando únicamente sus hojas de estilo.

En este proyecto se han utilizado las hojas de estilo externas, para definir por ejemplo los estilos de las fuentes, la disposición y tamaño de los distintos elementos, etc.

A continuación mostraremos un ejemplo del css utilizado para configurar la apariencia de la ventana de configuración del editor.

```
window.ventana{background-color:rgb(236,236,236);font-family:arial;font-size:13px;}
caption{background-color:rgb(236,236,236);font-weight:bold;text-align:center;color:rgb(119,163,197);}
description.titulo{font-weight:bold;font-size:1.5em;margin-left:375px;margin-top:5px;}
textbox.editor{width:50em;}
label.editor{background-color:rgb(236,236,236);font-weight:bold;text-align:center;color:rgb(119,163,197);}
```

Como podemos ver, en la primera línea definimos las propiedades para la ventana principal, que tendrá el atributo *class* 'ventana', aquí se define el color del fondo en el formato RGB(*Red, Green,Blue)*, la fuente que se utilizará, en este caso *Arial* y su tamaño en píxeles.

En la segunda línea se definen las propiedades para el *caption* o título, que son las nombradas anteriormente, pero en este caso también definiremos que la fuente deberá estar en 'negrita', esto se realiza mediante la propiedad *font-weight*.

En la tercera línea, definiremos las propiedades para el elemento *textbox* donde establecemos que su tamaño debe ser de 50em, lo que significa que ocupará un tamaño igual a 50 veces el tamaño de letra definida para el elemento.

Para finalizar en la última línea se definen las propiedades para la etiqueta editor, como el color de fondo, el alineamiento del texto, su peso (negrita) y su color.

5.1.6 Configuración regional (locale)

En este directorio se encuentran los ficheros DTD de la aplicación, aunque este tipo de ficheros se utilizan generalmente para definir la estructura y sintaxis de un archivo XML o SGML y de esta manera, mantener una consistencia entre todos los ficheros que lo utilicen, en este caso se ha usado únicamente para definir entidades con las etiquetas de los distintos componentes de la interfaz gráfica.

Una entidad [12] ofrece una entrada abreviada que se puede situar en el documento XML. El nombre abreviado es lo que se incluirá en el XML. Las entidades resultan muy útiles para repetir información o bloques grandes de texto que pueden estar guardados en archivos separados. El nombre abreviado va seguido de un punto y coma (;) en el documento XML (&abbName;)

Así, de esta manera tenemos todos los textos que se muestran al usuario contenidos en un único fichero por cada vista, y en el caso de querer traducir la extensión a otros idiomas, bastará con crear otros ficheros similares con los textos traducidos.

En este caso, los ficheros se encuentran dentro del subdirectorio *es-ES,* ya que esta extensión está escrita en el lenguaje Español, si quisiéramos por ejemplo tenerla traducida al inglés, bastaría con crear un subdirectorio *en-US* que contenga los mismos ficheros con los textos en este idioma.

Como ejemplo se mostrará el contenido del fichero administrador.dtd:

```
<!ENTITY  administrador.titulo "Administrar Diagramas de Flujo">
<!ENTITY  diagramas "Diagramas de Flujo del Usuario">
<!ENTITY  modificar "Modificar">
<!ENTITY  eliminar "Eliminar">
<!ENTITY  nuevo "Nuevo">
<!ENTITY  importar "Importar">
<!ENTITY  exportar "Exportar">
<!ENTITY  descripcion "Descripcion">
<!ENTITY  configurar "Configurar">
<!ENTITY etiqueta.url.incluidas "URLs asociadas">
```

Como se puede ver, por cada texto, hay un elemento ENTITY seguido por el nombre que se utilizará en el fichero XUL para referenciar dicho texto y a continuación el mensaje que se mostrará.

En el fichero XUL, la referencia se realizará en la propiedad que deseemos declarándola como: "*&nombre_texto;*", en esta extensión se ha utilizado, por ejemplo, para la etiqueta de un botón, mediante la siguiente línea:

```
<button id="importar_diagrama" label="&importar;"
onclick="importarDiagrama();" />
```

En nuestro caso como hemos visto, el nombre del texto contiene la sentencia "*Importar*", así en el caso de que quisiéramos traducir nuestra extensión al inglés sólo necesitaríamos otro fichero DTD con el texto "*Import*".

De esta manera vemos que es muy sencillo, traducir o modificar cualquier texto de la aplicación.

5.1.7 Capas XUL

Las capas XUL [13], son una manera de añadir un elemento a la interfaz de usuario en un documento XUL, durante el tiempo de ejecución. Una capa (en inglés *overlay*), es un archivo que marca elementos XUL para insertar en puntos de anclaje específicos, dentro del "documento maestro". Estos fragmentos indican que los elementos pueden ser añadidos, modificados o eliminados.

En el desarrollo de esta extensión se ha implementado un fichero XUL encargado de contener este elemento *overlay*. El fichero se llama *overlay.xul*, y se ha utilizado para integrar la

extensión presente a la interfaz gráfica del navegador *Firefox*. En concreto se utiliza para permitir que el menú principal de la extensión aparezca en el menú de Herramientas del navegador.

A continuación se irá explicando el contenido de dicho fichero:

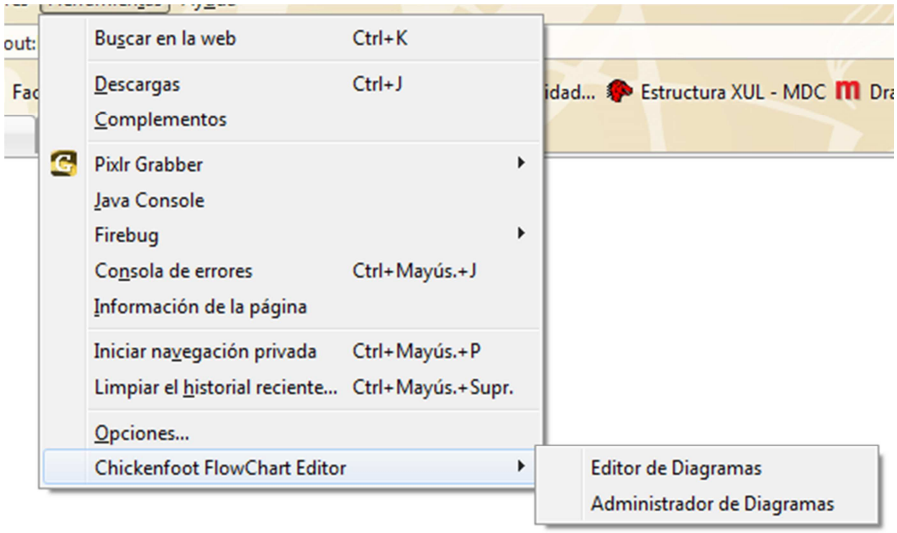
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE overlay SYSTEM "chrome://editor/locale/overlay.dtd">
<overlay id="helloworld-overlay" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <script type="text/javascript" src="utilidades.js"></script>
  <menupopup id="menu_ToolsPopup">
    <menu id="editor" label="&extension;">
        <menupopup id="menu_ToolsPopup2">
            <menuitem id="editor" label="&editor;" oncommand="abrirEditor();"/>
            <menuitem id="gestor" label="&gestor;" oncommand="abrirAdministrador();"/>
        </menupopup>
    </menu>
 </menupopup>
</overlay>
```

En la primera parte del contenido del fichero se puede comprobar cómo se empieza con la definición del tipo de documento tratado aquí, es decir, se comienza definiendo el elemento *overlay*, junto con sus propiedades, y se indica que se trata de un fichero de tipo XUL.

Después se encuentra la etiqueta *overlay*, que define este tipo de fichero y posteriormente se indican el fichero *JavaScript* que se utilizará, en este caso, para abrir las distintas pantallas que forman la extensión (el editor y el administrador).

En este menú sólo utilizaremos las funciones para abrir las dos vistas principales de la extensión, por lo que sólo necesitaremos el *JavaScript* nombrado y descrito anteriormente y que es el encargado de mostrar estas vistas.

Dentro del elemento *overlay* se define un menú simple (*menupopup*) con dos entradas (*menuitem*). El id de este menú es *menu_ToolsPopup*, lo que significa que su contenido se añadirá a la ventana que tenga un elemento similar con el mismo identificador, que en nuestro caso será en el menú Herramientas del navegador.



5.1.8 Versiones de Firefox válidas

Como se ha explicado anteriormente, en el fichero install.rdf se indican las versiones mínima y máxima del *Mozilla Firefox* que soporta la extensión. En este caso la versión mínima queda establecida por el uso de la nueva API para realizar el *drag and drop* de los elementos del *canvas*, por lo que será la versión de *Firefox* 3.5 (*Gecko* 1.9.1) y posteriores.

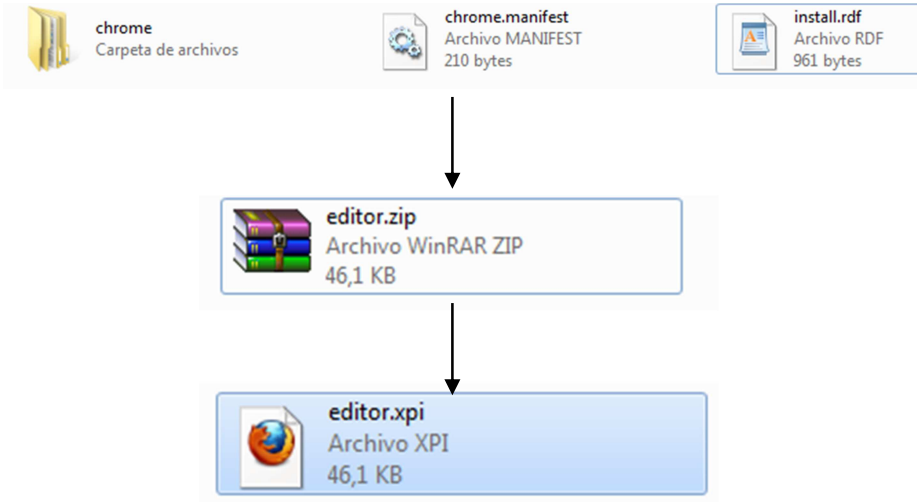
```
<em:targetApplication>
  <Description>
    …
    <em:minVersion>3.5</em:minVersion>
    <em:maxVersion>3.6.*</em:maxVersion>
  </Description>
</em:targetApplication>
```

Esto significa que la extensión podrá ser instalada en un navegador *Firefox* que se encuentre dentro de este rango de versiones. Este rango se va a especificar siguiendo el siguiente criterio: poner como versión mínima aquella con la que el desarrollador se compromete a que todo funcionará correctamente, y poner como versión máxima una no superior a la máxima que esté disponible. En el momento en que se escribe esta memoria, la máxima versión completa disponible para descargar es la versión 4.0.1, sin embargo, como ya se ha comentado anteriormente, en el desarrollo de este proyecto se ha utilizado la versión 3.6.13 .

5.1.9 Cómo crear el paquete de instalación XPI

Una vez que se han descrito todos los ficheros que forman la extensión, se procederá a indicar cómo se crea e instala el fichero XPI.

Como ya se ha visto anteriormente, el instalable contendrá los directorios *content*, *skin* y *locale*, además de los ficheros de configuración *install.rdf* y *chrome.manifest*.
Mediante el uso de una utilizad ZIP, empaquetaremos estos ficheros obteniendo un fichero denominado editor.zip, para convertirlo en un ejecutable de *Firefox* bastará con renombrar la extensión a .xpi.



Para su instalación en *Firefox* bastará con abrir este paquete con el navegador.

5.1.10 Aplicación final

Además de todos los archivos de configuración e instalación de la extensión, esta generará una serie de ficheros propios para obtener los datos de los diagramas creados.

Una vez se ha instalado la extensión, se creará una carpeta llamada *chifled* dentro del directorio del perfil de *Firefox* [14].

El directorio donde se encuentra alojado el perfil en *Mozilla Firefox*, dependerá del sistema operativo que se utilice:

- **En Windows Vista/Windows 7**: C:\Users\<Windows login/user name>\AppData\Roaming\Mozilla\Firefox\Profiles\<Nombre del Perfil>\
- **En Windows 2000/XP**: C:\Documents and settings\[usuario]\Datos de programa\Mozilla\Firefox\Profiles\[aleatorio].[perfil]\
- **En Windows 95/98/ME**: C:\WINDOWS\Application Data\Mozilla\Firefox\Profiles\[aleatorio].[perfil]\
- **En GNU/Linux**: /home/[usuario]/.mozilla/firefox/[aleatorio].[perfil]
- **En Mac OS X**: ~/Library/Mozilla/Firefox/Profiles/<Nombre del Perfil>/ o ~/Library/Application Support/Firefox/Profiles/<Nombre del Perfil>/

Dentro de este directorio de la extensión estará el fichero de configuración.

Este fichero se llama config.xml y es el que define el editor de texto a utilizar para abrir el código de los diagramas. El fichero tendrá un contenido similar a este:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<editor>textarea</editor>
```

Dentro de la etiqueta editor, estará la ruta del ejecutable del editor de texto elegido, en este caso se utilizará el *textarea* para mostrar el código.

Además por cada diagrama creado, se generará un directorio con el nombre de dicho diagrama. Estos directorios tendrán cada uno, un fichero XML con el nombre del diagrama, en el que se encontrará toda la información sobre éste. En este fichero se describirá cada elemento del diagrama y su situación dentro del *canvas* así como los ficheros de código asociados y las URLs asociadas a dicho diagrama.

Este fichero tendrá la siguiente estructura en función de sus elementos:

```
<?xml version="1.0"?>
<diagrama nombre=" " descripcion=" ">
<caja id="">
  <x></x>
  <y></y>
  <ancho></ancho>
  <alto></alto>
  <flechaO></flechaO>
  <texto></texto>
  <ficheroAsociado></ficheroAsociado>
    </caja>
<flecha id="">
  <etiqueta></etiqueta>
  <inicioX></inicioX>
  <inicioY></inicioY>
  <finX></finX>
  <finY></finY>
  <puntoCentroX></puntoCentroX>
  <puntoCentroY></puntoCentroY>
  <elementoOrigen></elementoOrigen>
  <tipoOrigen></tipoOrigen>
  <elementoDestino></elementoDestino>
  <tipoDestino></tipoDestino>
</flecha>
<elipse id="">
  <x1></x1>
  <y1></y1>
  <x2></x2>
  <y2></y2>
  <ancho></ancho>
  <texto></texto>
  <urlAsociada></urlAsociada>
</elipse>
 <fin id="">
  <x></x>
  <y></y>
  <flechaD></flechaD>
 </fin>
 <url></url>
</diagrama>
```

El DTD que validará este fichero será el siguiente:

```
<!DOCTYPE DIAGRAMA[
    <!ELEMENT diagrama (caja*,flecha*,elipse*,fin?,url*)>
    <!ELEMENT caja (x,y,ancho,alto,flechaO?,texto,ficheroAsociado?)>
    <!ELEMENT flecha
(etiqueta,inicioX,inicioY,finX,finY,puntoCentroX,puntoCentroY,elemento
Origen,tipoOrigen,elementoDestino,tipoDestino)>
    <!ELEMENT elipse (x1,y1,x2,y2,ancho,texto,urlAsociada?)>
    <!ELEMENT fin (x,y,flechaD?)>
    <!ELEMENT url(#PCDATA)>
    <!ELEMENT x (#PCDATA)>
    <!ELEMENT y (#PCDATA)>
```

```
    <!ELEMENT ancho (#PCDATA)>
    <!ELEMENT alto (#PCDATA)>
    <!ELEMENT flechaO (#PCDATA)>
    <!ELEMENT texto (#PCDATA)>
    <!ELEMENT ficheroAsociado (#PCDATA)>
    <!ELEMENT etiqueta (#PCDATA)>
    <!ELEMENT inicioX (#PCDATA)>
    <!ELEMENT inicioY (#PCDATA)>
    <!ELEMENT finX (#PCDATA)>
    <!ELEMENT finY (#PCDATA)>
    <!ELEMENT puntoCentroX (#PCDATA)>
    <!ELEMENT puntoCentroY (#PCDATA)>
    <!ELEMENT elementoOrigen (#PCDATA)>
    <!ELEMENT tipoOrigen (#PCDATA)>
    <!ELEMENT elementoDestino (#PCDATA)>
    <!ELEMENT tipoDestino (#PCDATA)>
    <!ELEMENT x1 (#PCDATA)>
    <!ELEMENT y1 (#PCDATA)>
    <!ELEMENT x2 (#PCDATA)>
    <!ELEMENT y2 (#PCDATA)>
    <!ELEMENT urlAsociada (#PCDATA)>
    <!ELEMENT flechaD (#PCDATA)>
    <!ATTLIST diagrama nombre CDATA #REQUIRED>
    <!ATTLIST diagrama descripcion CDATA #IMPLIED>
    <!ATTLIST caja id ID #REQUIRED>
    <!ATTLIST flecha id ID #REQUIRED>
    <!ATTLIST elipse id ID #REQUIRED>
    <!ATTLIST fin id ID #REQUIRED>
]>
```

Como se puede ver, cada elemento llevará una información asociada, por ejemplo en el caso de las cajas el fichero determinará su posición mediante las coordenadas x e y en el *canvas*, el ancho y alto de esta, los identificadores de las flechas origen y destino, el texto que contendrá y el fichero de código asociado. Además al final se indicarán las URLs para las que se debe ejecutar el código final del diagrama.

Cada vez que se cree un diagrama, se creará una carpeta dentro del directorio del perfil de *Firefox*, el fichero de configuración del editor y el fichero con la información de todos los elementos del diagrama. Además por cada caja habrá un fichero con el código introducido, este fichero tendrá el nombre de la caja y extensión chk.

Una vez esté el diagrama completo y cerrado, con todas las cajas unidas y la última al punto final, al hacer clic en Guardar, se crearán todos los ficheros descritos anteriormente, y además se generará automáticamente el fichero codigo.chk que contendrá el *script* final que se ejecutará al introducir las URLs definidas.

Capítulo 6

Pruebas

Tras el desarrollo de la extensión, esta fue sometida a un banco de pruebas para comprobar su correcto funcionamiento y detectar posibles errores. Además se comprobó que cumple correctamente con los requisitos enumerados en el capítulo 3. A continuación se describirán las pruebas llevadas a cabo sobre la extensión presentada.

6.1 Instalación de la extensión Chickenfoot

Se han realizado las siguientes pruebas para comprobar el requisito número 3: *La extensión deberá comprobar al inicio de su ejecución si está instalado el complemento de Chickenfoot y, en caso contrario, se mostrará un mensaje indicando la URL desde la cual se podrá descargar.*

- Lanzar la extensión sin instalar *Chickenfoot*.
- Lanzar la extensión habiendo instalado *Chickenfoot* pero sin reiniciar el navegador.
- Lanzar la extensión habiendo instalado *Chickenfoot* y reiniciando el navegador.
- No aceptar la instalación de *Chickenfoot* y lanzar la extensión.
- Lanzar la extensión desactivando *Chickenfoot*.
- Lanzar la extensión con el complemento de *Chickenfoot* instalado y activo.

6.2 Administrador de diagramas

Se realizaron las siguientes pruebas para la comprobación del correcto funcionamiento del administrador y el cumplimiento de los requisitos asociados (4 y 20).

- Creación de un nuevo diagrama.
- Borrado de un diagrama.
- Exportación de un diagrama.
- Importación de un diagrama creado.

- Asociación de URLs a los diagramas y posterior comprobación de que se ejecuta el código correspondiente al introducir la URL en el navegador.

6.3 Editor de Diagramas

Se realizaron las siguientes pruebas para comprobar el correcto funcionamiento del editor de diagramas. Además se comprueba que se cumplen los requisitos asociados a este módulo (5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 21 y 22).

- Arrastrar y soltar los distintos elementos en el *canvas* central.
- Se comprueba que se solicita el identificador para el nuevo diagrama al abrir el editor.
- Se intenta crear un nuevo diagrama sin asignarle un identificador.
- Se intenta crear un diagrama con un identificador duplicado en el sistema.
- Se comprueba que en el título de la ventana del editor aparece el nombre del diagrama que se está modificando.
- Se observan todas las herramientas nombradas en el requisito 9.
- Se añade código a un proceso haciendo doble clic sobre una caja.
- Se establece un editor de texto instalado en el PC.
- Se establece como editor el *textarea*.
- Se abren varios *textareas* de código simultáneamente.
- Se intenta añadir varias flechas a un proceso.
- Se arrastran los elementos dentro del *canvas* y se comprueba que se ve la imagen de arrastre.
- Se selecciona un elemento haciendo clic sobre este.
- Se borra un elemento, seleccionándolo y pulsando suprimir.
- Se borran todos los elementos a través del botón Borrar.
- Se observa que se ofrece al usuario un identificador único dentro del diagrama al añadir un nuevo elemento.
- Se intenta añadir un elemento con un identificador duplicado.

- Se sitúa el ratón sobre una fuente de datos y se comprueba que aparece la URL asociada en una ventana emergente de información.
- Se modifica el identificador de un elemento, seleccionándolo y pulsando la tecla control.
- Se usan en código los datos de entrada de un proceso mediante el identificador asociado a la flecha correspondiente en el diagrama.
- Se crea un diagrama complejo.
- Se comprueba que al hacer doble clic sobre una caja compleja aparecen las opciones de ver el *script* final y abrir el nuevo diagrama en el editor.
- Se guarda un diagrama incompleto y se comprueba que aparece el mensaje de error correspondiente.

6.4 Aplicaciones finales de ejemplo.

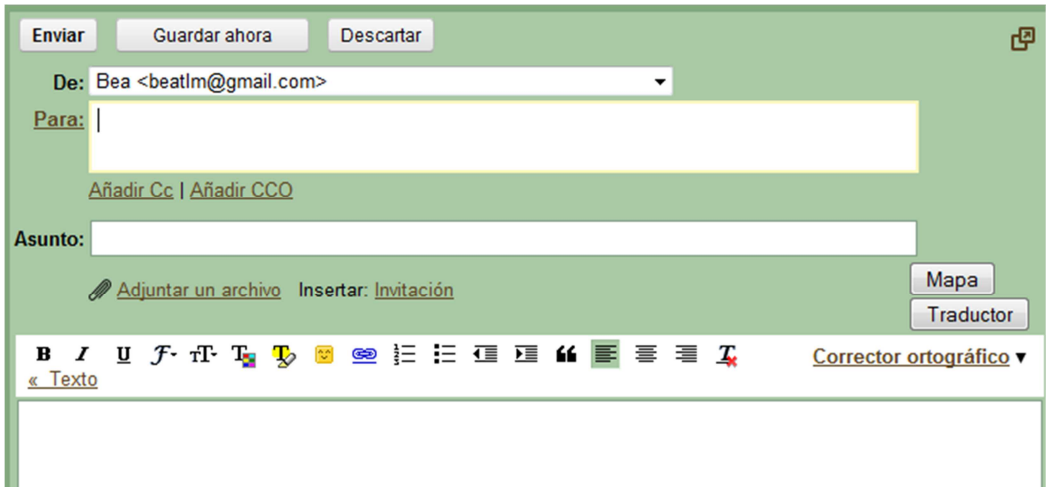
La mejor manera de comprobar el correcto funcionamiento de la extensión desarrollada es realizar una aplicación final con el editor de diagramas. Para ello se han desarrollado dos aplicaciones de ejemplo:

- Botón de búsqueda en Wikipedia:

  Se ha desarrollado una aplicación que añade un botón de búsqueda a la página del buscador Google. Este botón buscará el término introducido en el campo de búsqueda directamente en Wikipedia.

- Botón de traductor y mapas en Gmail

  Esta aplicación consiste en añadir dos botones con funcionalidades nuevas a la ventana de redactar nuevo mensaje en Gmail.



El botón 'Mapa' buscará en el texto del mensaje el patrón:
<map> Dirección </map>
Cogerá la dirección escrita, realizará una búsqueda en *Google maps* y posteriormente insertará en el mensaje un enlace al mapa obtenido.

El botón 'Traductor' buscará en el texto del mensaje la cadena:
<translate lan=idioma> Texto</translate>
Traducirá utilizando la herramienta *Google* traductor el texto pasado al idioma seleccionado en el atributo lan, posteriormente sustituirá el texto en el mensaje por el resultado traducido.

Capítulo 7

Conclusiones

En este capítulo se presentan las conclusiones a las que se ha llegado tras la realización del presente proyecto. Durante el desarrollo del mismo, se han encontrado diversos problemas que se han tenido que resolver, investigando y profundizando sobre las distintas tecnologías que se han utilizado finalmente, obteniendo así una visión más clara sobre cada uno de los aspectos que han intervenido.

Por un lado se ha profundizado en el mundo de las extensiones para el navegador *Mozilla Firefox*. Se ha podido comprobar que el desarrollo de este tipo de aplicaciones, gracias al lenguaje para interfaces gráficas XUL y a su acoplamiento con otras tecnologías como son *JavaScript* y CSS, es muy potente. De esta manera se puede afirmar que es posible desarrollar múltiples diseños y funcionalidades que son añadidas al propio navegador.

Sin embargo, a pesar de ser tan potente y flexible, su desarrollo no ha sido sencillo, ya que se ha requerido mucho esfuerzo para conseguir los resultados pretendidos para cumplir con los requisitos iniciales de este proyecto. Ha sido necesario tomar decisiones tanto en la implementación de la lógica funcional, como en el diseño de las interfaces gráficas del usuario.

Otra complicación ha sido el uso del elemento *canvas,* ya nombrado anteriormente, y su interacción con los eventos *drag & drop*. Esto se ha debido a que esta etiqueta, pertenece a una nueva implementación de HTML con lo que no es muy abundante la información y los ejemplos sobre esta materia. Se ha comprobado, cómo a pesar de las mejoras importantes que se ofrecen en esta nueva revisión, su uso aún no está muy extendido.

Y como última muestra de dificultad asociada al proyecto, estaría toda la lógica de programación que hay detrás de la aplicación, la cuál puede ser usada tanto por un desarrollador de aplicaciones finales, al cual se le ha tenido que ofrecer toda la funcionalidad que puede necesitar, como por un usuario final, ofreciéndole a este simplicidad en las operaciones de instalación de aplicaciones finales y su transparencia en cuanto a la lógica que hay por detrás.

Con todo esto, se puede concluir que los objetivos propuestos antes de empezar con el desarrollo del proyecto, han sido cumplidos de manera satisfactoria. Se ha logrado realizar una aplicación que ofrece la funcionalidad necesaria para el desarrollo de aplicaciones finales de una forma gráfica, permitiendo de este modo ampliar la funcionalidad de la extensión de *Chickenfoot*.

Las aplicaciones creadas mediante el uso del editor desarrollado permiten a los usuarios enriquecer su navegación por la web mediante la personalización y automatización de tareas de una forma sencilla gracias al lenguaje *Chickenfoot* elegido para la implementación de las aplicaciones finales.

Anexo A

Manual de instalación

En este apéndice se van a describir detalladamente los pasos para instalar la extensión desarrollada para el navegador *Mozilla Firefox*.

Dado que esta extensión depende de otra (*Chickenfoot*), se describirá también el proceso de instalación de esta última.

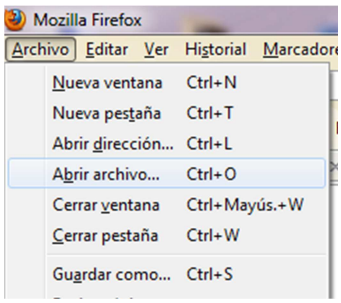
A.1 Instalación de la extensión ChickenfootFlowChartEditor

Para proceder a la instalación de esta extensión, necesitaremos como se ha nombrado anteriormente el paquete XPI instalable (en este caso, el fichero editor.xpi).

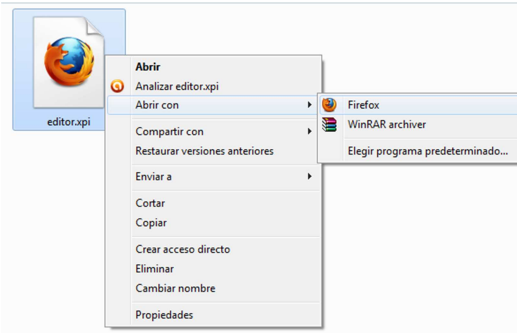
Será necesario abrir dicho fichero con el navegador *Firefox*.

Esto se puede realizar de dos maneras:

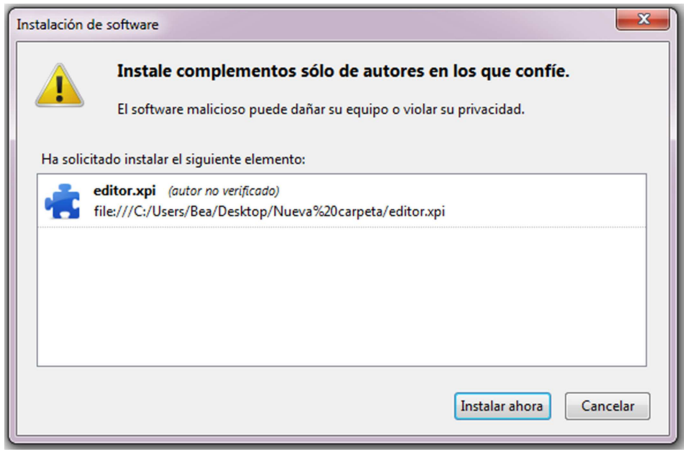
1. A través del navegador, abriendo el paquete mediante el menú **Archivo→Abrir Archivo** y seleccionando el fichero .xpi:



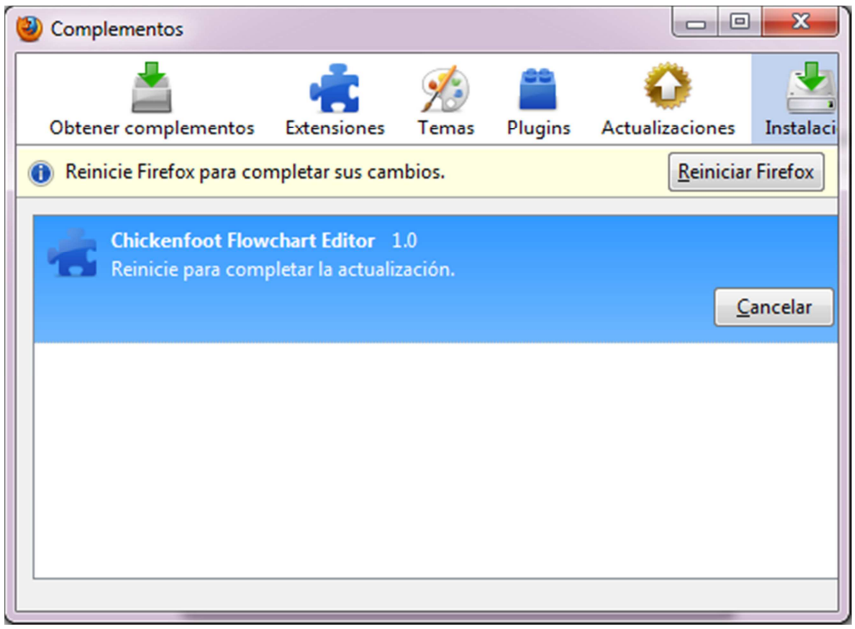
2. Pinchando directamente en el fichero con el botón derecho y ejecutando la opción **Abrir con →Firefox**.



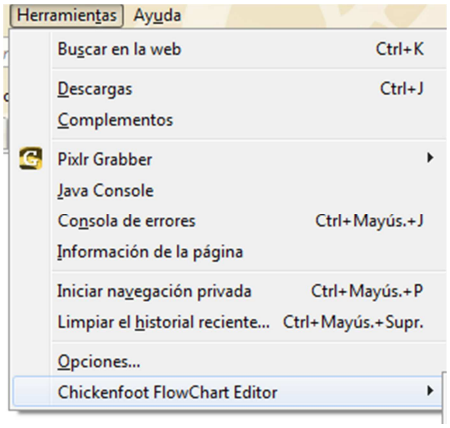
Una vez hecho esto, se abrirá la ventana del instalador de software del navegador. Esta ventana informará al usuario sobre el proceso de instalación, aportando datos como el nombre del fichero que se quiere instalar, y desde dónde se va a acceder a él.



Para continuar con la instalación deberemos pulsar el botón ***Instalar ahora***, a continuación se instalará la extensión y cuando se complete el proceso, se abrirá una ventana indicando que para realizar los cambios será necesario reiniciar el navegador, entonces se pulsará en el botón ***Reiniciar Firefox*** para continuar con la instalación.



Una vez reiniciado el navegador, podemos comprobar que la extensión se ha instalado correctamente porque en el menú de Herramientas aparecerá la opción *ChickenfootFlowChartEditor*



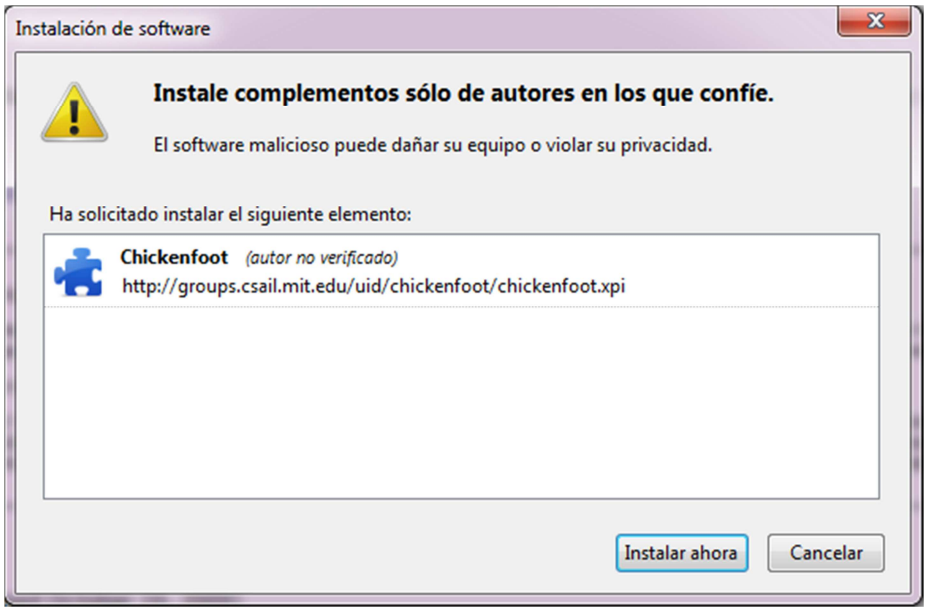
A.2 Instalación de la extensión Chickenfoot

Tras la instalación y una vez reiniciado el navegador, la extensión estará disponible para su uso. Lanzaremos la aplicación desde el menú de ***Herramientas→ChickenfootFlowChartEditor*** eligiendo cualquiera de las dos opciones disponibles, Administrador de diagramas o Editor de diagramas

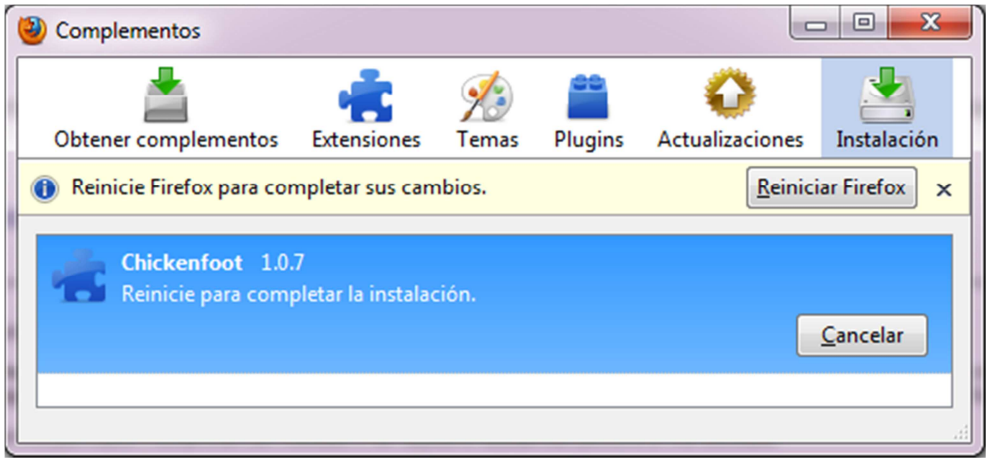
Después de elegir una opción, se abrirá la ventana correspondiente pero al ser la primera vez que se lanza la aplicación, aparecerá la siguiente alerta indicando que para poder utilizar la extensión, será necesaria la instalación del complemento Chickenfoot.

Para poder instalar el complemento, iremos a la dirección indicada y desde ahí pulsaremos sobre la opción ***Install Chickenfoot.***

Una vez hecho esto, se abrirá la ventana del instalador de software del navegador. Al igual que en el caso anterior se informará al usuario, indicando cuál es el nombre del fichero que se quiere instalar, y desde dónde se va a acceder a él.



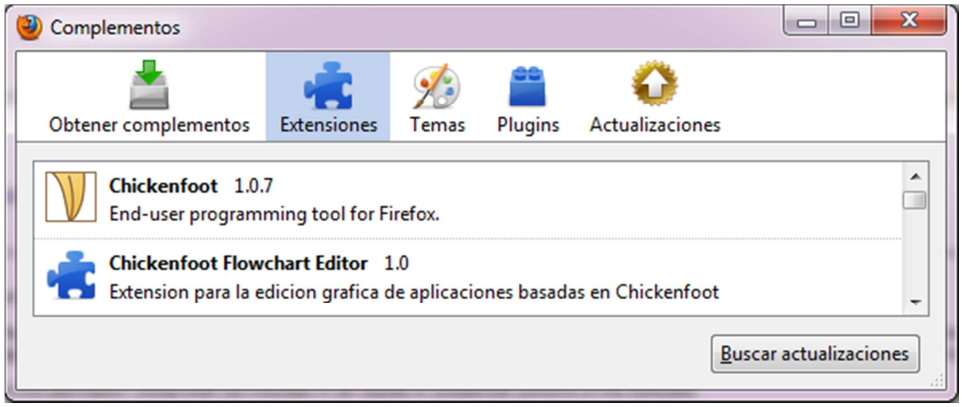
Para continuar con la instalación deberemos pulsar el botón ***Instalar ahora***, a continuación se instalará la extensión y cuando se complete se abrirá una ventana indicando que para realizar los cambios será necesario reiniciar el navegador.



Una vez reiniciado el navegador, ya disponemos de todo lo necesario para empezar a utilizar la extensión. Se puede comprobar que todo se ha instalado correctamente mirando en la ventana que muestra la información sobre los complementos instalados en el navegador.

Para ello accederemos al menú ***Herramientas→Complementos***, y ahí buscaremos las extensiones instaladas recientemente.

Deben aparecer las dos extensiones que hemos instalado previamente; ChickenfootFlowChartEditor y Chickenfoot.



Anexo B

Manual de usuario

En este apéndice se va a explicar el manejo y funcionamiento de la extensión *ChickenfootFlowChartEditor*, con el objetivo de facilitar su utilización por parte del usuario que quiera desarrollar aplicaciones *Chickenfoot* usando dicha extensión.

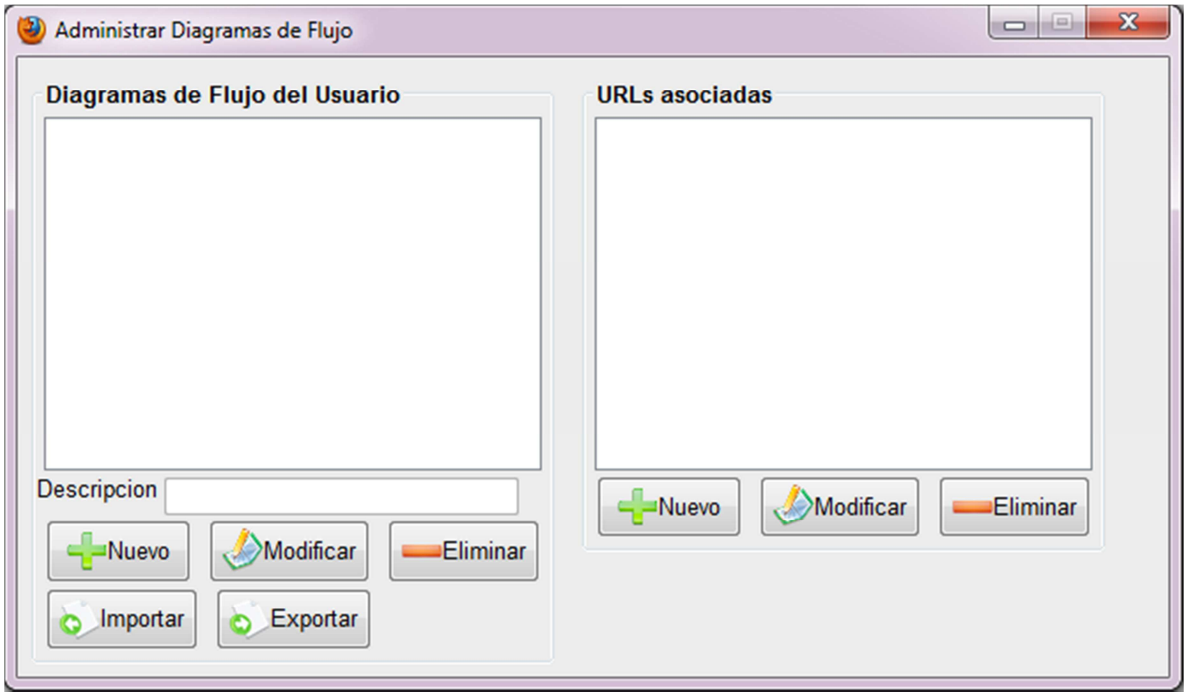
Se explicará en detalle el funcionamiento de las dos ventanas principales de la aplicación, el Administrador de Diagramas y el Editor de Diagramas, así como los distintos pasos que hay que seguir para crear un diagrama desde cero y obtener el *script* final, que será el ejecutado en el navegador.

Además, durante la explicación de los distintos pasos, se irá mostrando un ejemplo práctico que permita entender mejor el funcionamiento de la aplicación.

B.1 Administrador de diagramas.

La ventana principal de la aplicación y de la cual partiremos para crear nuestro diagrama, es el administrador. Accederemos a esta vista desde el menú ***Herramientas→ChickenfootFlowChartEditor→Administrador de diagramas***.

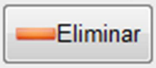
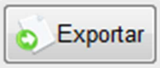
Al abrirla, obtendremos una ventana como la siguiente:



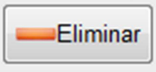
Como se puede observar, esta pantalla tiene dos partes, en la izquierda tenemos las opciones para cada diagrama, mientras que en la derecha tenemos las opciones para las URLs asociadas a los distintos diagramas.

Al ser la primera vez que se accede a la aplicación todos los campos están vacíos. Una vez hayamos creado un diagrama, en la parte izquierda aparecerán tanto el nombre como la descripción de cada diagrama, mientras que en la derecha aparecerá una lista con las URLs asociadas para las que se ejecutará el código final del diagrama seleccionado.

- Las opciones disponibles para los diagramas son:

Botón	Función
	Abrirá el editor para crear un diagrama nuevo.
	Abrirá el editor con el diagrama seleccionado cargado para su modificación.
	Eliminará el diagrama seleccionado y todos los ficheros asociados, la eliminación de un diagrama será permanente.
	Permite seleccionar un diagrama creado e importarlo a nuestra aplicación para proceder a su uso posterior.
	Permite exportar los ficheros de un diagrama creado.

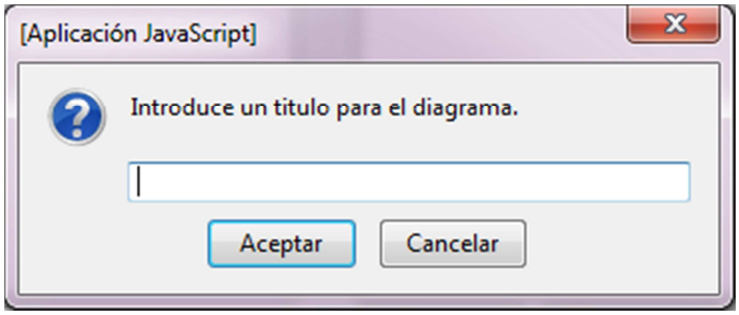
- Mientras que las opciones de las URLs son:

Botón	Función
	Añadir una URL para la que se ejecutará el diagrama seleccionado.
	Modificar la URL seleccionada.
	Eliminar la URL seleccionada .

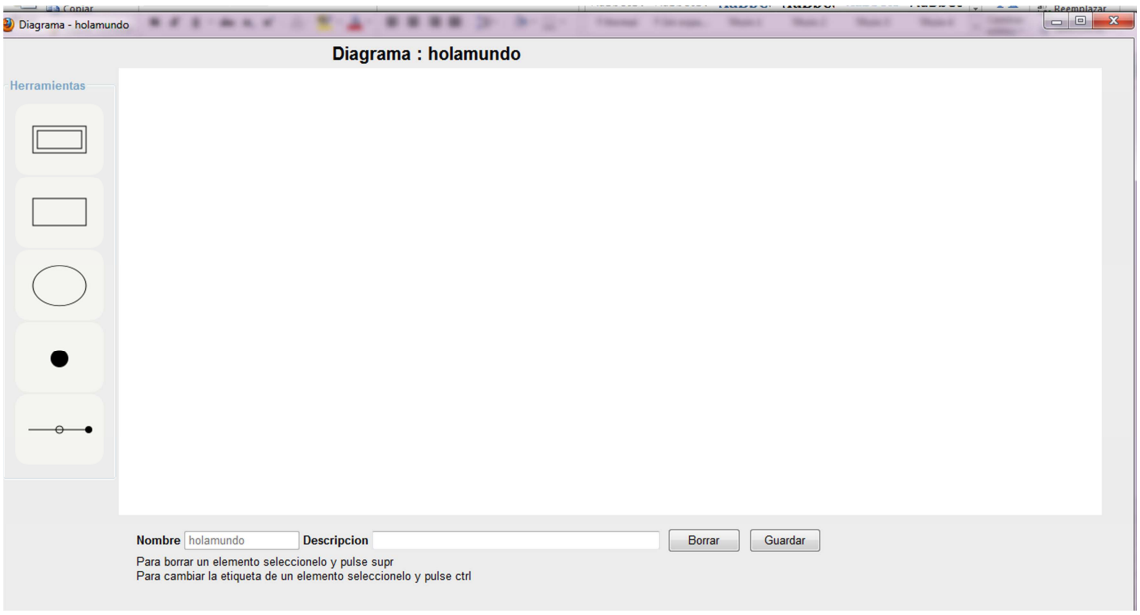
B.2 Editor de diagramas.

Se puede acceder a la vista del editor de diagramas, bien desde el menú de ***Herramientas→ChickenfootFlowChartEditor→Editor de Diagramas*** o a través de la ventana del administrador, pulsando sobre el botón Nuevo.

Al acceder al editor, aparecerá una ventana donde introduciremos el nombre del diagrama.



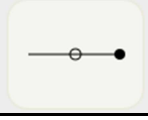
Después aparece la vista del editor vacía para crear el nuevo diagrama.



En la izquierda tenemos la barra de herramientas con todos los elementos para crear el diagrama.

Desde esta barra se irá arrastrando cada elemento hasta el canvas para ir añadiendo los procesos a nuestro diagrama.

Los elementos disponibles son:

Elemento	Descripción
	Proceso complejo
	Proceso sencillo
	Fuente de datos
	Punto final del diagrama
	Flecha de unión

Un proceso complejo será aquel que podrá contener otros procesos sencillos o complejos en su interior, de esta manera se permite ampliar la complejidad de los *scripts* finales.

Un proceso sencillo tendrá un código, unos datos de entrada (opcionales) y unos datos de salida.

La fuente de datos contendrá una URL desde la que se podrá cargar datos para manejarlos dentro del diagrama.

La flecha de unión, conectará procesos entre sí, o procesos con fuentes de datos, o el último proceso con el punto final del diagrama, representan el flujo de datos entre procesos.

Además todo diagrama deberá tener un punto de cierre o punto final y este deberá ser único.

Por último, en la parte inferior tenemos el nombre del diagrama, su descripción, que se podrá modificar desde esta vista, y varios botones:

- Borrar: Para eliminar todo el diagrama creado.

- Guardar: Para guardar el diagrama, además se comprobará si todos los elementos están unidos entre sí y estos a un punto final, y si todos los procesos contienen código, de esta forma se creará el script final correctamente, en caso contrario se mostrará una ventana de error.

Además como se indica en la parte inferior del editor, podemos borrar un elemento seleccionándolo y pulsando la tecla suprimir.

También es posible modificar una etiqueta de un elemento seleccionándolo y pulsando la tecla ctrl.

B.3 Creando un diagrama

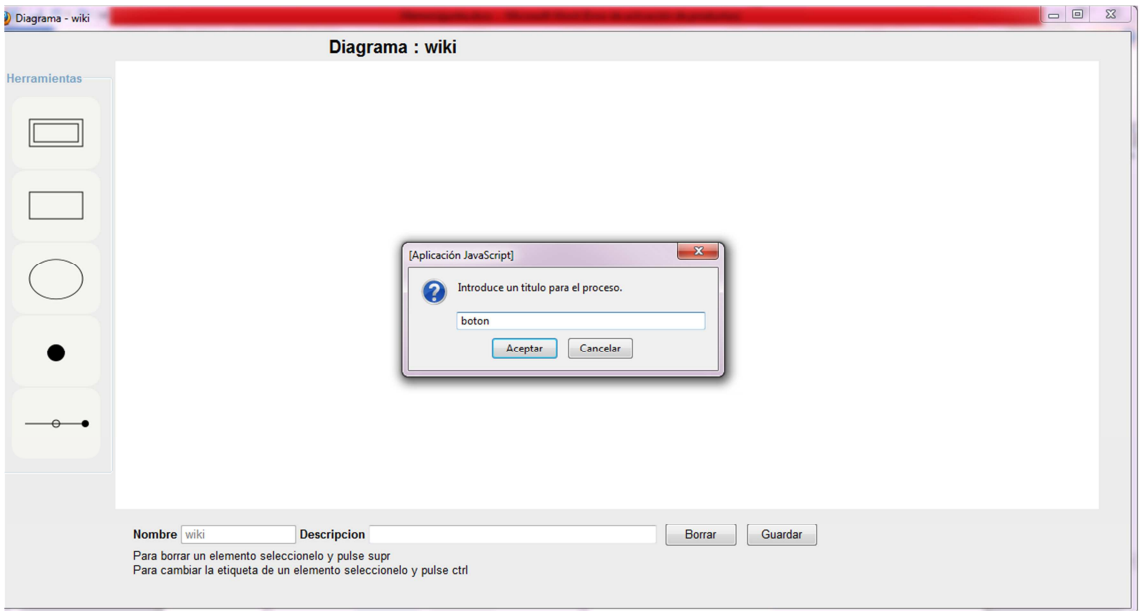
Vamos a crear un diagrama sencillo que añada un botón en la página del buscador *Google* y que realice la búsqueda e*n Wikipedia*.

A continuación se detallarán los pasos para crear este diagrama desde cero.

El primer paso será acceder al editor de diagramas, para ello iremos desde el menú ***Herramientas→ChickenfootFlowChartEditor→Editor de Diagramas.***

El siguiente paso será establecer un nombre para el diagrama.

Una vez estamos en el editor, arrastraremos la primera caja al canvas central y le daremos un identificador.

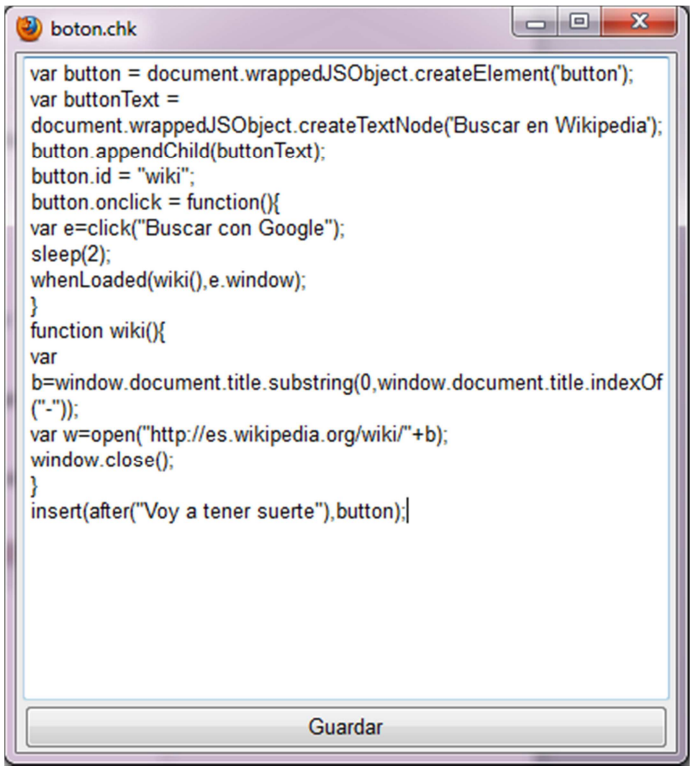


Para introducir el código en el lenguaje *ChickenFoot*, que será el que procese nuestra aplicación, haremos doble clic en la caja creada.

Entonces, se abrirá una ventana para elegir el editor con el que queremos editar el código a partir de ahora, si no tenemos ninguno instalado , podemos darle a cancelar y entonces se abrirá un textarea.

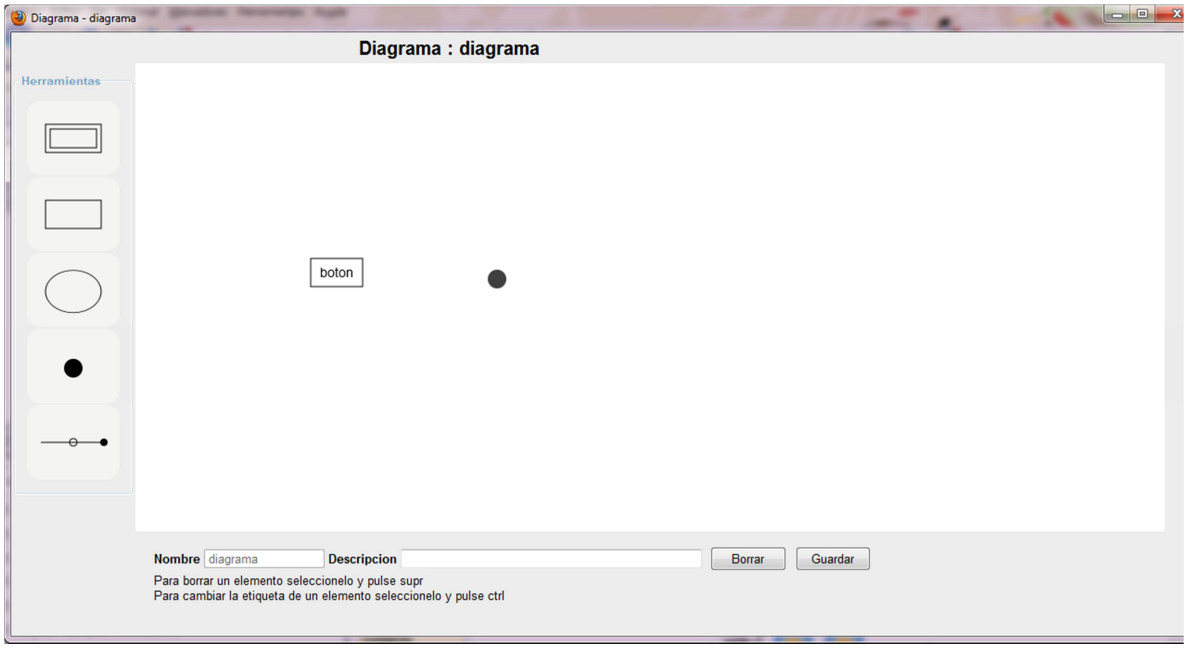
En esta ventana podemos introducir nuestro código, en la parte superior de la ventana podemos ver el nombre del proceso que estamos modificando.

Con el siguiente código, conseguiremos mostrar un botón al lado del botón “Buscar” de Google que al pulsarlo realizará la búsqueda en la página de Wikipedia.



Para continuar con nuestro diagrama, almacenaremos el código anterior pulsando el botón ***Guardar***.

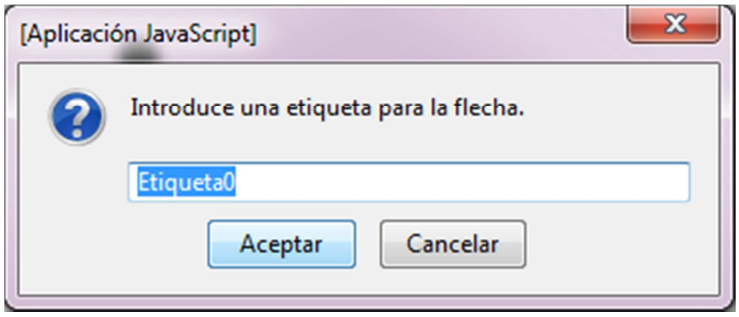
Una vez tenemos nuestro proceso, introduciremos el punto final del diagrama arrastrándolo desde la barra de herramientas.



Y para finalizar uniremos el proceso con el punto final usando la flecha, para ello haremos clic sobre el elemento de la barra de herramientas, y a continuación seleccionaremos el primer elemento, que será el proceso, y el destino, que será el punto final.

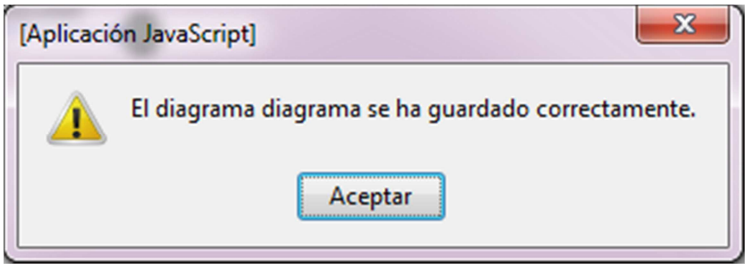
Aparecerá una ventana para introducir el identificador de la flecha, dejaremos el que viene por defecto.

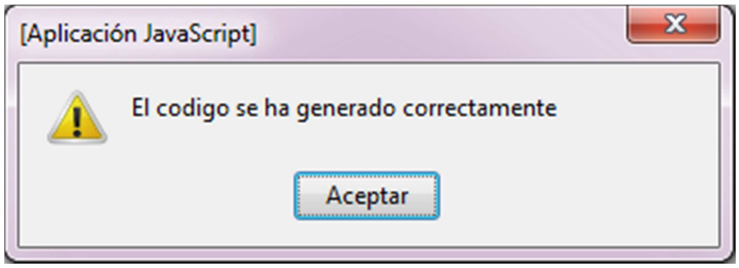
Si vamos a tener más procesos en el diagrama y necesitamos que estos se pasen datos, podemos identificar mediante la flecha los datos de salida del primer proceso y usarlos en el segundo mediante el identificador que le hayamos asignado.



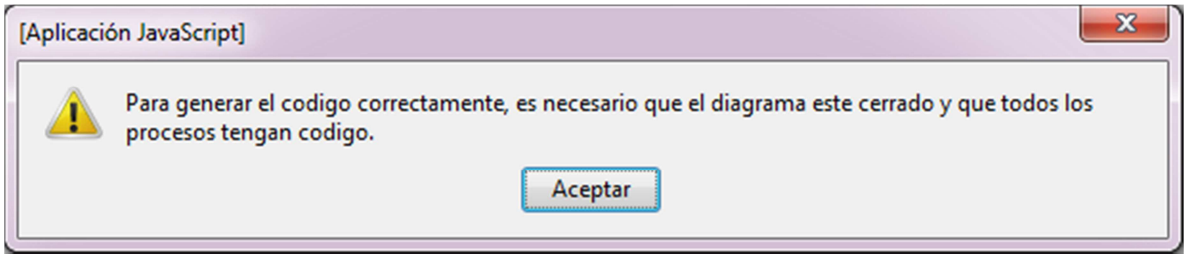
Una vez tenemos nuestro diagrama terminado, bastará con pulsar sobre el botón Guardar para generar el *script* final.

Si todo ha ido correctamente, es decir, si hemos añadido código al proceso y nuestro diagrama está cerrado con un punto de fin, nos mostrará un mensaje indicando que se ha guardado correctamente.



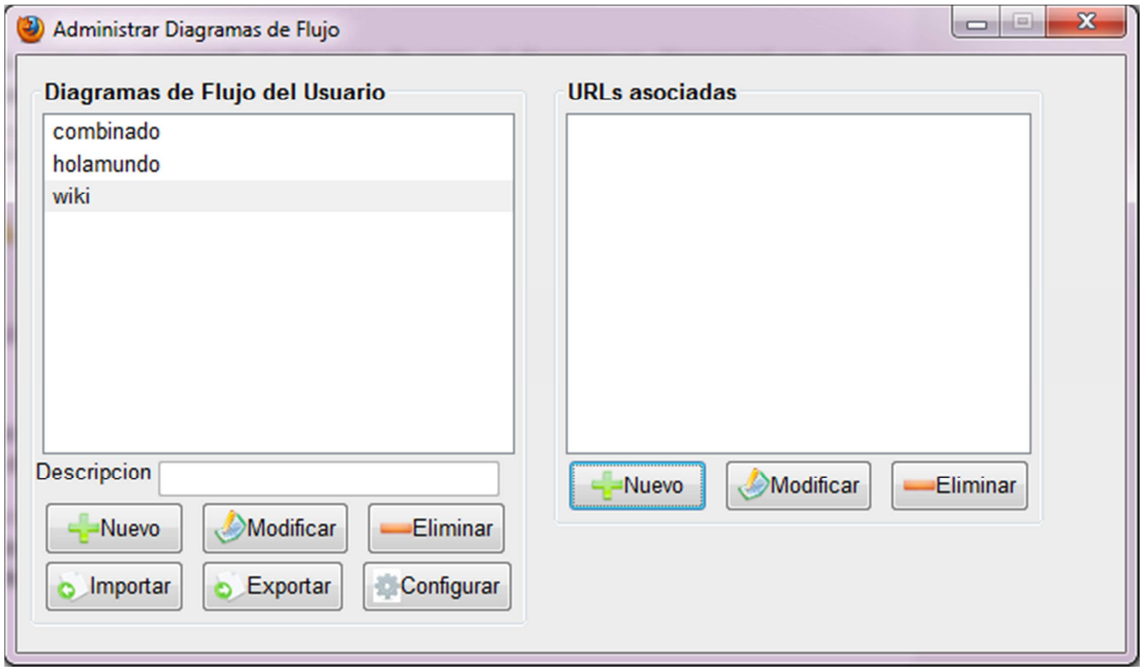


En caso contrario, aparecerá un mensaje de error, el diagrama se almacenará para poder modificarlo posteriormente, pero no se generará el código final.

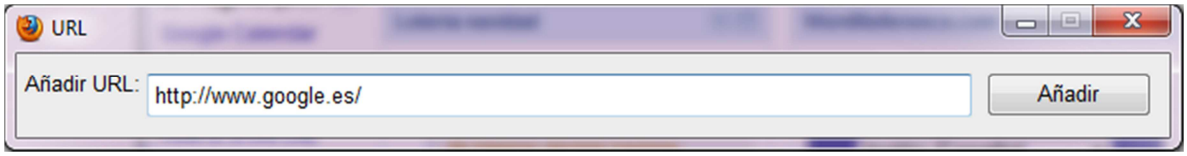


Una vez hemos guardado y se ha generado el código correctamente, ya tenemos nuestro primer diagrama creado, ahora solo faltará indicarle cuándo se tiene que ejecutar.

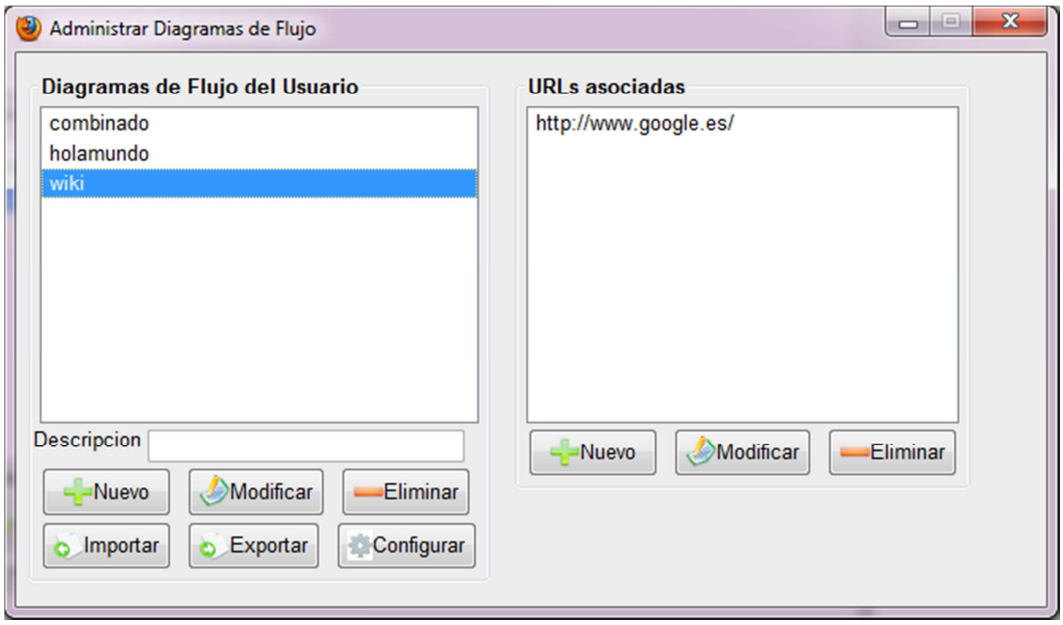
Esto se realizará asignándole una URL en la ventana del administrador de diagramas, para ello seleccionaremos el diagrama en la lista de la izquierda y pulsaremos sobre el botón Nuevo del menú de URLS.



Aparecerá una ventana donde podemos añadir la URL, en nuestro caso la de Google y pulsaremos en el botón Añadir.



Ya tenemos todo lo necesario, nuestro diagrama y una URL, así cuando se introduzca la URL en el navegador se ejecutará el *script* final que se ha creado mediante esta aplicación.



Para ver el resultado, introduciremos la URL de Google en el navegador, ahí podemos ver que hemos añadido un botón nuevo.

Anexo C

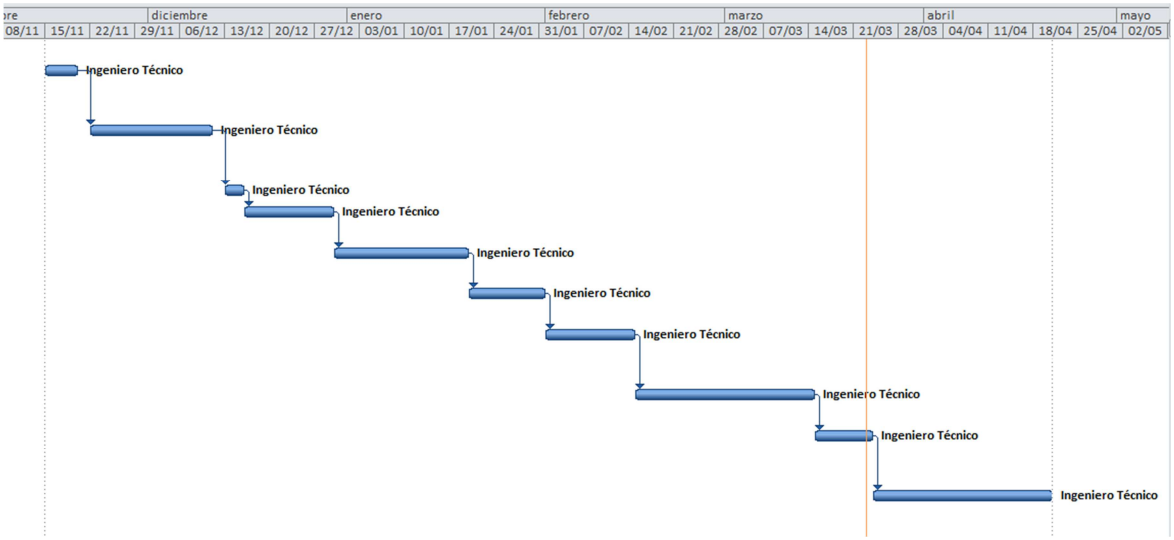
Planificación

En este anexo se muestra el coste total de desarrollo del proyecto junto con la planificación temporal del mismo.

En la siguiente página aparecen especificadas las tareas que se han llevado a cabo durante la realización del proyecto y el diagrama de *Gantt* correspondiente. Por cada tarea se detalla su descripción, duración, fechas de comienzo y fin y el identificador de las tareas que la preceden.

Según la estimación el proyecto comienza el día 15 de Noviembre del 2010 y finaliza el 20 de Abril del 2011 con lo que tiene una duración de 5 meses y 5 días.

A continuación se mostrarán los gastos detallados y ordenados según sean gastos asociados al personal, a los equipos o a otros costes directos Todos estos costes nos proporcionan el coste final cuyo valor total es de 64.675,83€.



Id de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	ESTUDIO INICIAL	20 días	lun 15/11/10	vie 10/12/10	
2	Consulta documentación inicial sobre el proyecto.	5 días	lun 15/11/10	vie 19/11/10	
3	Consulta documentación tecnologías	15 días	lun 22/11/10	vie 10/12/10	2
4	ESPECIFICACIÓN REQUISITOS	3 días	lun 13/12/10	mié 15/12/10	
5	Creación lista requisitos.	3 días	lun 13/12/10	mié 15/12/10	3
6	ANÁLISIS	10 días	jue 16/12/10	mié 29/12/10	
7	Análisis de la aplicación de las tecnologías	10 días	jue 16/12/10	mié 29/12/10	5
8	DISEÑO	15 días	jue 30/12/10	mié 19/01/11	
9	Diseño general de la aplicación	15 días	jue 30/12/10	mié 19/01/11	7
10	IMPLEMENTACIÓN	38 días	jue 20/01/11	lun 14/03/11	
11	Implementación interfaces gráficas.	8 días	jue 20/01/11	lun 31/01/11	9
12	Implementación funcionalidad del administrador	10 días	mar 01/02/11	lun 14/02/11	11
13	Implementación funcionalidad del editor	20 días	mar 15/02/11	lun 14/03/11	12
14	PRUEBAS	7 días	mar 15/03/11	mié 23/03/11	
15	Pruebas y documentación de resultados.	7 días	mar 15/03/11	mié 23/03/11	13
16	DOCUMENTACIÓN	20 días	jue 24/03/11	mié 20/04/11	
17	Memoria del proyecto	20 días	jue 24/03/11	mié 20/04/11	15

1.- Autor:

Beatriz López Moreno

2.- Departamento:

Informática y Telecomunicaciones

3.- Descripción del Proyecto:

Título	Extensión Firefox para la edición gráfica de aplicaciones basadas en Chickenfoot
Duración (meses)	**5**
Tasa de costes indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):

64.675,83 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Nombre de tarea	Trabajo real	Coste
Consulta documentación inicial sobre el proyecto.	40 horas	2.200,00 €
Consulta documentación tecnologías	120 horas	6.000,00 €
Creación lista requisitos.	24 horas	1.200,00 €
Análisis de la aplicación de las tecnologías	80 horas	4.000,00 €
Diseño general de la aplicación	120 horas	6.000,00 €
Implementación interfaces gráficas.	64 horas	3.200,00 €
Implementación funcionalidad del administrador	80 horas	4.000,00 €
Implementación funcionalidad del editor	160 horas	8.000,00 €
Pruebas y documentación de resultados.	56 horas	2.800,00 €
Memoria del proyecto	128 horas	8.000,00 €

EQUIPOS

Descripción	Coste	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Ordenador PC Intel	1.000,00	100	5	60	70,83

OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Costes imputable
Gastos fungibles (electricidad, material…)	200,00

RESUMEN DE COSTES

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	45.400
Amortización	70,83
Costes de funcionamiento	200
Costes Indirectos	9.137
I.V.A (18%)	9.868
Total	**64.675,83**

Acrónimos

API (*Application Programming Interface*)

CSS (*Cascading Style Sheets*)

DOM (*Document Object Model*)

DTD (*Document Type Definition*)

HTML5 *(HyperText Markup Language, version 5*)

RDF (*Resource Description Framework*)

SGLM (*Standard Generalized Markup Language*)

URI (*Uniform Resource Identifier*)

W3C (*World Wide Web Consortium*)

WHATWG (*Web Hypertext Application Technology Working Group*)

XBL (*XML Binding Language*)

XML (*Extensible Markup Language*)

XUL (*XML-based User-interface Language*)

Bibliografía

[1] XML

Apuntes de la asignatura Laboratorio Software de Comunicaciones08

[2] XML in 10 points

<http://www.w3.org/XML/1999/XML-in-10-points> [Consulta: 9 de noviembre de 2010]

[3] HTML-5

<http://www.desarrolloweb.com/articulos/que-es-html5.html> [Consulta: 5 de noviembre de 2010]

[4] Tutorial Canvas

<https://developer.mozilla.org/en/canvas_tutorial> [Consulta: 5 de noviembre de 2010]

[5] CSS

<http://www.w3c.es/divulgacion/guiasbreves/hojasestilo> [Consulta: 9 de noviembre de 2010]

[6] JavaScript

<http://www.w3schools.com/js/js_intro.asp> [Consulta: 9 de noviembre de 2010]

[7] Características de Firefox

<http://www.mozilla-europe.org/es/firefox/features/> [Consulta: 25 de octubre de 2010]

[8] Drag and Drop

<https://developer.mozilla.org/En/DragDrop/Drag_and_Drop> [Consulta: 7 de noviembre de 2010]

[9] XUL

<https://developer.mozilla.org/es/XUL> [Consulta: 4 de noviembre de 2010]

<http://es.wikibooks.org/wiki/XUL/Introducci%C3%B3n> [Consulta: 4 de noviembre de 2010]

[10] Creando una extensión

<https://developer.mozilla.org/es/Creando_una_extensi%C3%B3n> [Consulta: 21 de abril de 1999]

[11] Chickenfoot

<http://www.webmonkey.com/2010/02/get_started_with_chickenfoot/> [Consulta: 10 de noviembre de 2010]

<http://groups.csail.mit.edu/uid/chickenfoot/index.php> [Consulta: 10 de noviembre de 2010]

[12] Entidades DTD

<http://msdn.microsoft.com/es-es/library/ms256483%28v=vs.80%29.aspx> [Consulta: 10 de noviembre de 2010]

[13] Capas XUL

<https://developer.mozilla.org/es/Creando_una_extensi%C3%B3n> [Consulta: 10 de noviembre de 2010]

[14] Perfil Firefox

<http://www.mozillaes.org/documentacion/index.php?title=Perfil_%28Mozilla_Firefox%29 > [Consulta: 10 de noviembre de 2010]